Vysoká škola báňská – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Sběr systémových a síťových informací a jejich následné zpracování pomocí nástroje RRDTool

System and Network Information Capture and Processing with RRDTool

2015 Bc. David Jucha

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Zadání diplomové práce

Student: **Bc. David Jucha**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2601T013 Telekomunikační technika

Téma: Sběr systémových a síťových informací a jejich následné zpracování pomocí nástroje RRDTool
System and Network Information Capture and Processing with RRDTool

Zásady pro vypracování:

Cílem diplomové práce je naprogramovat vhodné skripty, které budou sbírat a následně vykreslovat různé systémové a síťové informace pomocí nástroje RRDTool.
1. Popis protokolu SNMP.
2. Skriptovací jazyky bash, perl, python.
3. Návrh řešení sběru informací a jejich automatické zpracování.
4. Ověření funkčnosti navrženého řešení.
5. Vytvoření vhodných návodů pro praktickou výuku.

Seznam doporučené odborné literatury:

Douglas Mauro, Kevin Schmidt, *Essential SNMP* O'Reilly Media 2005, ISBN-13: 978-0596008406
David N. Blank-Edelman, *Automating System Administration with Perl: Tools to Make You More Efficient* O'Reilly Media 2009, ISBN-13: 978-0596006396

Podle pokynů vedoucího diplomové práce

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Pavel Nevlud**

Datum zadání: 01.09.2014

Datum odevzdání: 07.05.2015

doc. Ing. Miroslav Vozňák, Ph.D.
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2015 ..... David Jucha .....

Na tomto místě bych rád poděkoval všem, kteří mi poskytli cenné rady a informace, potřebné k dokončení mé diplomové práce, ať už se jednalo o spolužáky nebo mé známé a kamarády.

Především bych zde rád poděkoval mému vedoucímu, panu Ing. Pavlu Nevludovi, který si vždy ochotně našel čas na osobní a pro mě velmi přínosné konzultace. Také mu děkuji za cenné rady, nápady a připomínky, které spolu s mým úsilím dovedly tuto diplomovou práci do aktuální podoby.

V neposlední řadě bych rád poděkoval mým blízkým rodinným příslušníkům za podporu a umožnění studia na VŠB-TU Ostrava.

Abstrakt

Tato diplomová práce se zabývá sběrem systémových a síťových informací a jejich následným zpracováním pomocí nástroje RRDtool. Cílem práce bylo naprogramovat vhodné skripty pro sběr informací ve vhodném jazyce. V práci jsem se zaměřil na základní popis: protokolu SNMP, potřebného pro sběr informací z počítačové sítě; skriptovacích programovacích jazyků Bash, Perl a Python; nástrojů Cron a Lm-sensors. Detailněji jsem se věnoval nástroji RRDtool a jeho hlavním funkcím Create, Update a Graph. V další části práce jsou popisovány příkazy použité ve vytvořených skriptech v jazyce Perl pro získávání informací z lokálních stanic (s využitím nástrojů Lm-sensors nebo nástrojů, vestavěných v OS Ubuntu), síťových stanic nebo síťových zařízení (v obou případech s využitím protokolu SNMP). V závěrečné kapitole jsou vytvořeny návody pro praktickou výuku v laboratořích.

Klíčová slova: RRDtool, SNMP, Bash, Perl, Cron, Lm-sensors, skripty

Abstract

This thesis deals with the collection of system and network information and its processing using RRDtool. The aim of the thesis was to develop suitable scripts to collect information in the appropriate language. At thesis, I focused on the basic description of: SNMP protocol needed to collect information from a computer network; script languages Bash, Perl and Python; Cron and Lm-sensors tools. I have dealt with in more detail RRDtool and its main functions Create, Update and Graph. In another part of this thesis are described commands used in the created Perl scripts to collect information from local stations (using the tools of Lm-sensors or tools built into OS Ubuntu), networked stations or network devices (in both cases using SNMP). In the final chapter are created tutorials for practical training in laboratories.

Keywords: RRDtool, SNMP, Bash, Perl, Cron, Lm-sensors, scripts

Seznam použitých zkratek a symbolů

AMD	–	Advanced Micro Devices
APT	–	Advanced Packaging Tool
Bash	–	Bourne Again Shell
CD	–	Compact Disc
CF	–	Consolided Function
DCE	–	Data Circuit-Terminating Equipment
DS	–	Data Source
DST	–	Data Source Type
HD	–	High Definition
HW	–	Hardware
IETF	–	Internet Engineering Task Force
IOS	–	Internetwork Operating System
IP	–	Internet Protocol
IPv6	–	Internet Protocol, Version 6
Lm-sensors	–	Linux monitoring sensors
MIB	–	Management Information Base
MTU	–	Maximum Transmission Unit
NaN	–	Not a Number
NAS	–	Network Attached Storage
OID	–	Object Identifier
OS	–	Operating System
PC	–	Personal Computer
PHP	–	Hypertext Preprocessor
PID	–	Process Identifier
RAM	–	Random Access Memory
RFC	–	Request For Comment
RGB	–	Red-Green-Blue
RIP	–	Routing Information Protocol
RRA	–	Round Robin Archive
RRD	–	Round Robin Database
RRDtool	–	Round Robin Database Tool
SGMP	–	Simple Gateway Management Protocol
SNMP	–	Simple Network Management Protocol

SSH	–	Secure Shell
SW	–	Software
TCP/IP	–	Transmission Control Protocol/Internet Protocol
Telnet	–	Telecommunications Network
UDP	–	User Datagram Protocol
UPS	–	Uninterruptible Power Supply
URL	–	Uniform Resource Locator
UTC	–	Coordinated Universal Time
VLAN	–	Virtual Local Area Network
VPN	–	Virtual Private Network
XFF	–	Xfiles Factor

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

Úvod

S rozvojem počítačů, serverů, síťových prvků a zařízení, využívajících síťové připojení a následně i připojení k internetu, vyvstávají požadavky na jejich monitorování. Monitorování může být potřebné z mnoha důvodů - ať už se jedná o zajištění stability těchto zařízení nebo jen získávání různých informací o jejich HW. To může být zajímavé především pro správce serverů nebo poskytovatelů internetového připojení, kteří mají potřebu vytvářet statistiky, obsahující třeba provoz na rozhraních jejich síťových prvků, ze kterých mohou vyvozovat různé závěry.

Pro získávání informací a správu sítě byl přímo vytvořen protokol SNMP, kterému je věnována úvodní kapitola této diplomové práce. Protokol SNMP je v práci využíván asi v polovině praktických měření.

V další kapitole jsou zmíněny základní vlastnosti skriptovacích programovacích jazyků, bez kterých by nebylo možné vytvářet automatické skripty pro sběr různých informací. V této kapitole je také přiblíženo základní použití nástroje Cron, sloužícího pro automatické spouštění skriptů, bez kterého bychom se při sběru dat také neobešli. Abychom mohli získávat informace o teplotách, napětích a rychlostech ventilátorů v počítači bylo potřeba se seznámit s nástrojem Lm-sensors, který nám umožní získávat tyto informace. Proto se zde věnujeme i nástroji Lm-sensors.

Švýcar Tobias Oetiker vytvořil nástroj RRDtool, který nám umožňuje vytvářet databáze, do nich ukládat informace a z nich vytvářet grafy. Základnímu popisu tohoto nástroje spolu s jeho třemi hlavními funkcemi - Create, Update a Graph je věnována další kapitola. Jednotlivé parametry těchto funkcí, které je potřeba zadávat při jejich volání, jsou popsány v příloze této práce. U každé z funkcí Create, Update a Graph je vždy uveden krátký příklad, který je umístěn v odpovídajících podkapitolách.

Další kapitola je věnována praktické části práce. Tato praktická část je rozdělena na tři oblasti monitorování různých informací v různých umístěních s využitím různých nástrojů nebo protokolu. V první části, která se dělí na dvě menší, jsou sbírány informace z lokální stanice (počítače, notebooku, serveru), kdy se nejprve využívá nástroj Lm-sensors a následně jen vestavěné nástroje OS Ubuntu. Druhou oblastí je monitorování stanic, zapojených v počítačové síti, kde je využíván protokol SNMP. Poslední oblast je zaměřena na sběr informací ze síťových zařízení (routeru a switche firmy Cisco), opět s využitím protokolu SNMP. V každé z těchto oblastí bylo vytvořeno několik skriptů pro sběr různých informací z těchto stanic/zařízení. Součástí této práce jsou i návody pro praktickou výuku, které odpovídají výše popsaným rozdělením a jsou umístěny v příloze.

Cílem této práce je čtenáři přiblížit použití nástroje RRDtool, seznámit jej s různými doplňkovými nástroji, potřebnými pro sběr a automatické získávání informací pomocí vytvořených skriptů. V praktické části je kladen důraz na to, jakým způsobem a pomocí jakých příkazu jsou informace ze systému stanice/zařízení získávány. V každém skriptu, odpovídajícímu jednotlivým měřením, jsou vždy vytvářeny tři výstupní grafy ve dvou velikostech. Jeden z grafů je umístěn v hlavní části práce, jeden dlouhodobější je umístěn v příloze. Všechny ostatní grafy, včetně skriptů pro jejich získávání spolu s vytvořenými databázemi, jsou umístěny v elektronické podobě na přiloženém CD.

1 SNMP

V dnešní době nastal veliký rozmach IP (Internet Protocol) sítí. Samotná síť neobsahuje jen PC (Personal Computer), routery, switche a servery, jak tomu bylo dříve. S aktuálními možnostmi informačních technologií a techniky samotné nastává trend internetu věcí. Aktuální sítě obsahují navíc mnoho dalších zařízení - notebooků, tiskáren, televizí, NAS (Network Attached Storage) serverů, UPS (Uninterruptible Power Supply) a v neposlední řadě i mobilních telefonů, tabletů, konvertibilních zařízení, hodinek atd. K tomu, abychom zajistili jejich bezproblémový běh, je potřeba je nejlépe neustále monitorovat. Nezajímá nás jen to, jestli jsou zapnutá, ale i spousta jiných informací, které nám mohou tato zařízení poskytovat. Takto poskytnuté informace můžeme dále vyhodnocovat a následně provádět případné změny na daném zařízení nebo síti. Přesně k těmto účelům slouží SNMP (Simple Network Management Protocol). Jeho vznik se datuje na rok 1988, kdy odpověděl na požadavky, volající po standardu pro řízení zařízení, komunikujících pomocí protokolu IP. SNMP poskytuje sadu operací, pomocí kterých můžeme tato zařízení vzdáleně monitorovat.

Základ SNMP tvoří jednoduchá sada operací, která poskytuje administrátorům možnost změnit stav zařízení, podporujících SNMP. Například můžeme pomocí SNMP vzdáleně vypnout dané rozhraní routeru, umístěného v síti, nebo můžeme monitorovat datový tok procházející tímto rozhraním. Mnoho síťových prvků umožňuje monitorovat jejich teplotní senzory, které nás v případě překročení určité teploty mohou opět pomocí SNMP informovat.

SNMP je ve velké míře zmiňován v souvislosti s routery, umožňujícími jejich správu, například pomocí Telnetu (Telecommunication Network) či SSH (Secure Shellu). Neměli bychom ale opomíjet, že jej lze využít na široké škále zařízení, využívajících IP síť. SMNP vychází ze SGMP (Simple Gateway Management Protocolu), který byl navržen právě pro správu routerů. U SNMP je oblast využití mnohem širší - může být využit na systémech Unix a Windows, tiskárnách, modemech, UPS atd. I SW (software), běžící na jakémkoliv zařízení, který umožňuje přijímat informace SNMP, může být řízen. SNMP tedy není vyčleněn pouze pro hardware HW (hardware), umožňuje řídit i SW, jako jsou například webové servery a databáze. [1]

1.1 Verze protokolu SNMP

IETF (Internet Engineering Task Force) je mezinárodní organizace seskupující dobrovolníky, která je zodpovědná i za definování standardů pro protokoly, související s internetovým provozem. Do této skupiny spadá i SNMP. IETF vydávají RFCs (Requests for Comments), které jsou specifikací pro mnoho protokolů. Při schvalování RFC existuje ucelený postup, kdy RFC mění svůj status - od navrhované normy až po standard RFC. A teď detailněji k tomuto postupu. IETF nejprve navrhne například daný protokol jako navrhovanou normu, na kterou mohou reagovat firmy či lidé různými připomínkami. Dalším krokem je vypracování normy s již implementovanými a schválenými připomínkami. Až dojde ke schválení nebo neschválení připomínek, je danému RFC přidělen status standard RFC. Abychom byli korektní, existují ještě statusy „historický“ a „experimentální“. His-

torický označuje tu normu, která již byla nahrazena normou novější. Experimentální pak označuje takovou normu, která ještě není připravená se stát standardem. Níže si stručně shrneme, co obsahují jednotlivé verze SNMP protokolu. [1]

1.1.1 SNMP Version 1

- Je definován v RFC 1157 a již je ve stavu „historický".
- Jeho bezpečnost je založena jen na „community stringu" - nejedná se o nic jiného, než heslo potřebné pro přístup k informacím daného zařízení, které nebylo při přenosu nijak zabezpečeno.

1.1.2 SNMP Version 2

- Je definován v RFC 3416, RFC 3417 a RFC 3418.
- Taktéž používá pro autentizaci „community string".
- Jedná se o dnes nejhojněji využívanou a podporovanou verzi.

1.1.3 SNMP Version 3

- Je definován v RFC 3410, RFC 3411, RFC 3412, RFC 3413, RFC 3414, RFC 3415, RFC 3416, RFC 3417, RFC 3418 a RFC 2576.
- Hlavním přínosem této verze je bezpečnost.
- Důraz je kladen na šifrování a autentizaci pomocí jména a hesla.

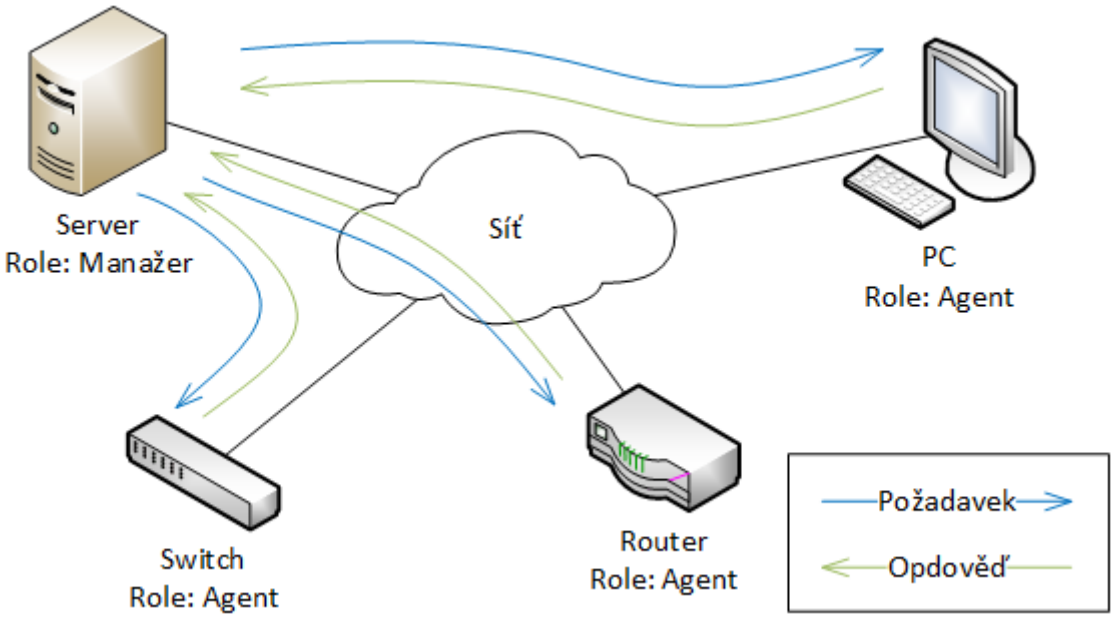
1.2 Entity SNMP

Při použití SNMP vystupují v dané komunikaci dvě entity - *manažer* a *agent*. *Manažer* je většinou server, na kterém běží určitý systém umožňující obstarávání sítě. Je zodpovědný za dotazování a získávání informací od *agentů* (routerů, switchů, serverů atd.), umístěných v síti. Tyto informace pak může dále analyzovat a zjišťovat tak případné problémy na daném zařízení.

Komunikace může také probíhat opačným směrem a to v případě „trap události". Pomocí této trap události může *agent* sdělit *manažerovi*, že se něco stalo. Trap události jsou, na rozdíl od výše uvedeného dotazování a odpovídání směrem od *manažera* k *agentovi*, asynchronní. To znamená, že neodpovídají na žádný dotaz, ale hlásí *manažerovi* určitou anomálii v síti. Na tu pak může *manažer* reagovat a provádět určitá opatření. Například router pošle *manažerovi* informaci o tom, že jedno z jeho rozhraní bylo vypnuto. Ten pak může reagovat a zjišťovat další okolnosti, proč se tak stalo.

Druhou vystupující entitou je *agent*. *Agent* je SW běžící na síťovém zařízení, které chceme monitorovat, a může mít dvě podoby: Buď jako samostatná SW aplikace nainstalovaná na daném zařízení, která může třeba běžet na pozadí jako démon nebo se může

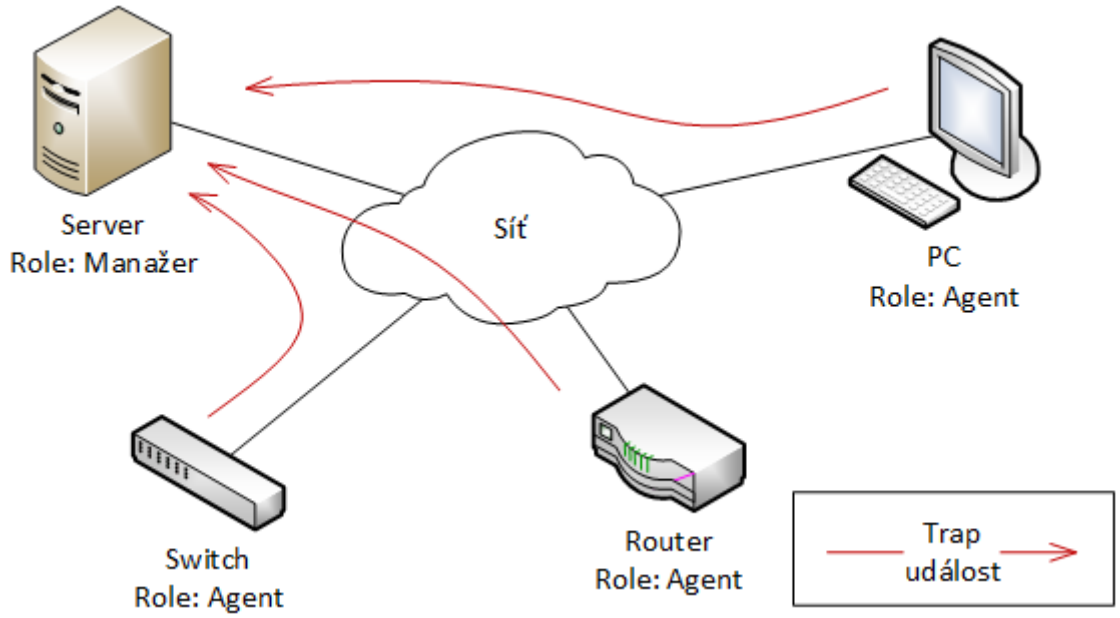
jednat přímo o zakomponovanou součást systému. Zde je příkladem IOS (Internetwork Operating System) na routerech a switchích společnosti Cisco. Dnes bývá ve většině IP zařízeních vestavěná nějaká verze SNMP *agenta*, jelikož je SNMP brán jako standard pro monitorování sítě. Tím výrobci zařízení také podstatně zjednodušují práci síťovým administrátorům. *Agent* má možnost si sledovat širokou škálu provozních informací o svém zařízení a tyto informace pak může poskytovat *manažerovi*. Ten může, jak již bylo výše zmíněno, reagovat na dotaz *manažera* (obrázek 1.1) nebo zaslat *manažerovi* trap událost (obrázek 1.2) o nějaké nečekané události, aby ji vyhodnotil. Některá zařízení navíc umožňují zasílat zprávu typu trap událost s obsahem „all clear“, kterou říkají *manažerovi*, že nečekaná událost, dříve ohlášená zprávou trap, je již v pořádku. Tato zpráva může být velice užitečná právě pro *manažera*, který se nemusí neustále dotazovat na aktuální stav, stačí mu totiž vyčkat na zprávu trap s obsahem „all clear“.



Obrázek 1.1: Komunikace mezi manažerem a agentem při použití zpráv typu „požadavek“ a „odpověď“

Názorným příkladem může být například situace, kdy je *agentem* router a *manažerem* síťový server. Router umožňuje monitorovat tok dat na jeho rozhraních. U jednoho z rozhraní spadne datový tok na nulu, což je neobvyklé, a tak zašle zprávu typu trap *manažerovi*. Ten zprávu vyhodnotí a začne zjišťovat, co se děje. Ještě během zjišťování mu přijde zpráva typu trap s obsahem „all clear“ a tím mu oznámí, že je vše v pořádku. *Manažer* si zde může oddechnout, že byl problém bez jeho zásahu vyřešen a že se o něm dověděl pomocí zprávy trap obsahující „all clear“.

Oba výše uvedené scénáře mohou probíhat v jeden okamžik, jelikož v této komunikaci neexistují žádná omezení. *Manažer* se může neustále dotazovat *agenta* na nějakou informaci a přitom mu může *agent* poslat zprávu trap o jiné anomálii v síti. [1]



Obrázek 1.2: Komunikace mezi manažerem a agentem při použití zprávy typu „trap událost“

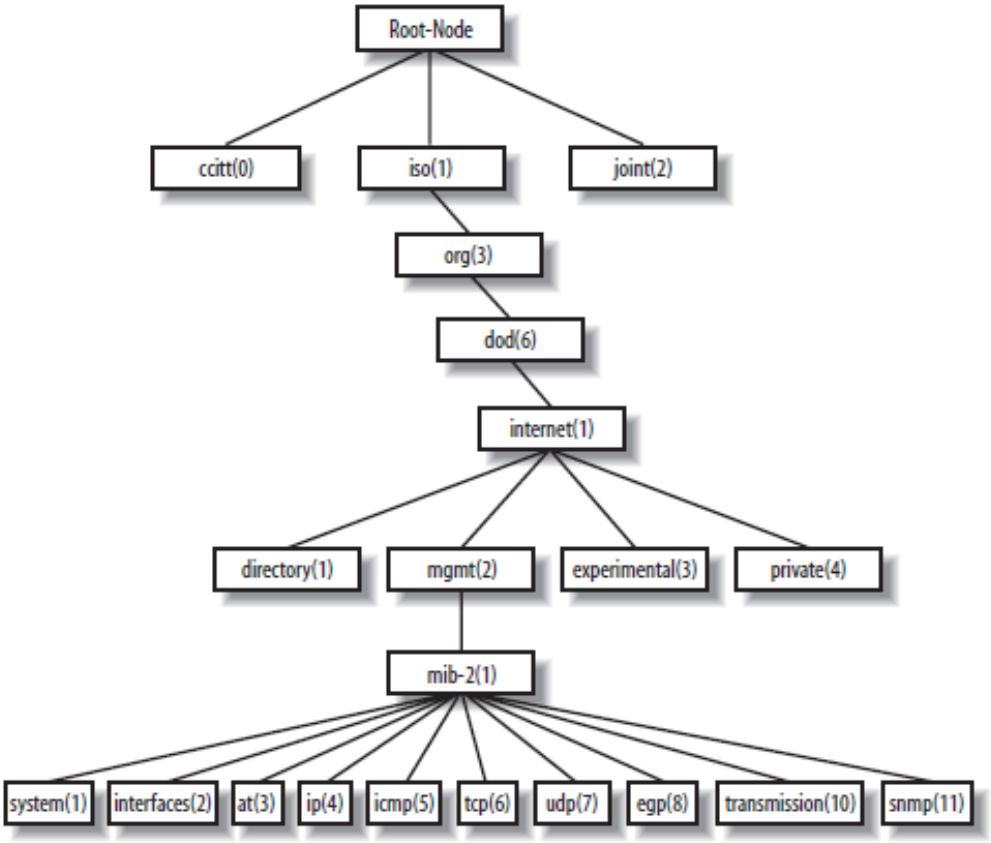
1.3 Management Information Base

MIB (Management Information Base) je definována pomocí objektů a jejich chováním. *Agent* obsahuje skupinu objektů, do kterých zaznamenává informace. Jeden z takových objektů je například provozní status rozhraní routeru. Ten může nabývat stavů „up“, „down“ nebo „testing“. Z tohoto objektu lze následně jednoduše zjistit, v jakém stavu je dříve uvedené rozhraní routeru.

MIB pak může být chápána jako databáze objektů, které *agent* zaznamenává. Jakýkoliv stav nebo statistická informace, kterou může *manažer* číst, je obsažena v této databázi. Každý *agent* může implementovat více takových databází a všichni *agenti* musí implementovat základní MIB nazvanou MIB-II definovanou v RFC 1213. Tento standard definuje proměnné pro věci, jako jsou například statistiky síťových rozhraní (jejich rychlosti, MTU (Maximum transmission unit), přijaté a odeslané pakety atd.) a také různé jiné věci vztahující se k systému samotnému. Hlavním úkolem MIB-II je poskytnout základní informace, týkající se protokolové rodiny TCP/IP (Transmission Control Protocol/Internet Protocol). Na základní strukturu MIB-II se můžeme podívat na obrázku 1.3.

Jaké další informace může být užitečné sbírat? Mimo základní MIB-II, kterou jsme zmiňovali výše, je definováno mnoho dalších MIB, zaměřených na určité oblasti. Každá takto existující MIB má svou definici v RFC. I každý výrobce si může definovat své vlastní MIB, ty jsou pak označovány jako „proprietární“ od daného výrobce. Abychom této problematice lépe porozuměli, uvedeme si zde další příklad.

Uvažujme, že výrobce síťových prvků uvede na trh nový model routeru. V něm bude vestavěný SNMP *agent*, který bude odpovídat na dotazy *manažerovi* nebo případně



Obrázek 1.3: Stromová struktura obsahující podstrom MIB-II [1]

mu bude posílat zprávy typu trap, ale pouze pro ty proměnné, které jsou definovány v základním standardu pro MIB-II. Tento router však bude pravděpodobně implementovat i jiné MIB, které budou pokrývat proměnné pro rozhraní, jež tento router má, například RFC 2115 pro Frame Relay. Kromě toho může mít tento router nějaké nové významné funkce a jejich informace, které stojí za sledování, ale nejsou pokryty žádnou standardní MIB. Proto má tento router implementovanou i svou vlastní proprietární MIB, která má za úkol umožnit monitorovat právě tyto nové významné funkce a jejich informace tohoto nového routeru. [1]

1.4 Správa zdrojů hostitele

Správa zdrojů hostitele (místa na disku, využití paměti, ...) je důležitou částí správy sítě. Rozdíl mezi tradiční systémovou administrací a správou sítě bychom dnes již těžce hledali. Také bychom zde mohli zmínit známou větu Johna Gage, v tu dobu zaměstnance firmy *Sun Microsystems*: „The Network is Computer“. Z té bychom mohli vyčíst, že není podstatné, jestli spravujeme lokální zařízení nebo zařízení na síti. MIB (RFC 2790) pro

správu zdrojů hostitele definuje sadu objektů, pomáhajících řídit kritické aspekty systémů Unix a Windows. Nemusí se jednat pouze o tyto systémy, nezáleží na systému, ale na tom, jestli daný systém umožňuje běh SNMP *agenta*. V této MIB můžeme naleznout například objekty, zahrnující kapacitu disků, počet uživatelů daného systému, počet spuštěných procesů, případně informace o nainstalovaném SW. Mnoho lidí se dnes spoléhá na webové stránky, poskytující různé služby. Aby si byli jisti, že servery, na kterých tyto stránky či služby běží, fungují správně, je důležité je monitorovat. Nemusí však stačit monitorovat pouze tyto servery, je potřeba se zaměřit také na routery a ostatní zařízení, která mohou běh a služby těchto serverů ohrozit. Bohužel některé implementace *agentů* pro tyto platformy neimplementují tyto MIB do té doby, než jsou požadovány. [1]

1.5 Object identifier

Objekty databáze MIB jsou organizovány do podoby stromové struktury. Z této struktury vychází i jejich pojmenování. OID (Object identifier) je složen z celých čísel, oddělených tečkami. Tato čísla odpovídají jednotlivým listům stromové struktury. Nemusíme však používat pouze čísla, existuje i forma přijatelnější pro člověka. Ta se skládá z celých jmen jednotlivých uzlů, oddělených tečkami. Záleží na každém, jaký způsob pojmenování objektů si vybere. Například OID 1.3.6.1.2.1 = iso.org.dod.internet.mgmt.mib-2. Oba způsoby jsou ekvivalentní. Na obrázku 1.3 je zobrazena stromová struktura databáze MIB spolu s názvy jednotlivých uzlů a jejich ekvivalentními číselnými hodnotami. Stromová struktura má jeden kořen (Root-Node). Jakákoliv další část, která má potomky, je nazývaná podstromem. Jestliže potomky nemá, je koncovou hodnotou - listem. [1]

1.6 SNMP v operačním systému Ubuntu

Pro zajištění a plné využití funkcionality protokolu SNMP je potřeba mít k dispozici minimálně jednoho *manažera* a jednoho *agenta*. Kdybychom šli do detailů, *manažer* i *agent* může běžet na stejném zařízení. V režimu `localhost` se dotazuje *manažer agenta* na informace i když jsou fyzicky na jediném zařízení. V praxi je ale více použitelná situace, kdy *manažer* a *agent* jsou na různých zařízeních v síti. Proto si v následujících podkapitolách ukážeme, jak nakonfigurovat *manažera*, respektive *agenta*. Jelikož testování probíhá na síti, využívající protokolu IPv6 (Internet Protocol Version 6), bude i návod uzpůsobený pro tento protokol.

1.6.1 Server (manažer)

Pro nainstalování SNMP na server v roli *manažera* postupujeme následovně:

- `$ sudo su` - získání práv uživatele ROOT
- `# apt-get update` - aktualizace databáze balíčků
- `# apt-get install snmp` - samotná instalace nástroje SNMP

Po této základní instalaci je *manažer* připraven k použití, k jeho používání již není potřeba provádět žádné další úkony. Pro ulehčení práce s OID si ještě můžeme nainstalovat

balíček `snmp-mibs-downloader`, který nám umožní získávat informace nikoliv na základě OID, ale na základě jména určeného pro dané OID. Instalaci provedeme obdobným způsobem jako při instalaci `snmp` s tím rozdílem, že řetězec `snmp` nahradíme řetězcem `snmp-mibs-downloader`.

1.6.2 Klient (agent)

U klienta respektive `agenta` je instalace o něco málo složitější. Nejprve provedeme samotnou instalaci a dále budeme pokračovat v editaci konfiguračního souboru. SNMP démona tedy nainstalujeme následujícím způsobem:

`$ sudo su` - získání práv uživatele ROOT

`# apt-get update` - aktualizace databáze balíčků

`# apt-get install snmpd` - samotná instalace SNMP démona

Po instalaci je potřeba přistoupit k editaci konfiguračního souboru SNMP démona například pomocí textového editoru NANO. Zadáme tedy:

```
# nano /etc/snmp/snmpd.conf
```

A ve výše umístěném konfiguračním souboru `snmpd.conf` zakomentujeme znakem # řádek obsahující:

```
agentAddress udp:127.0.0.1:161
```

Naopak zde odkomentujeme a upravíme řádek obsahující:

```
#agentAddress udp:161,udp6:[::1]:161
```

na řádek:

```
agentAddress udp:161,udp6:161
```

Další změnou je zakomentování řádku obsahujícího:

```
rocommunity public default -V systemonly
```

A přidání řádku pod něj, který bude obsahovat:

```
rocommunity6 public
```

Po ukončení editace souboru `snmpd.conf` soubor uložíme a editaci opustíme. V editoru NANO k tomu slouží kombinace zkratek „Ctrl-O“ a „Ctrl-X“. Před použitím SNMP démona je potřeba jej restartovat, to provedeme pomocí příkazu:

```
# service snmpd restart
```

Nyní je démon připraven k použití. Jeho funkcionalitu si můžeme ověřit tak, že na *manažerské* stanici zadáme příkaz:

```
# snmpwalk -v 2c -c public udp6:[ipv6_adresa_agenta]
```

Po potvrzení tohoto příkazu by se nám na obrazovce měl vypsat seznam všech možných informací, které můžeme získat ze stanice `agenta`.

2 Skriptovací programovací jazyky a použité nástroje Cron a Lm-sensors

Programovací jazyky můžeme obecně rozdělit do dvou kategorií. Buďto máme jazyky kompilované a nebo skriptovací. Zdrojový kód námi vytvořeného programu pro zpracování pomocí skriptovacího jazyka se nazývá skript. Tyto skripty jsou zpracovávány speciálním programem, kterému se říká interpret. Interpret překládá skript až v okamžiku jeho spuštění. Zde je hlavní rozdíl oproti kompilovaným programovacím jazykům. V kompilovaném programovacím jazyce vytvoříme zdrojový kód, který následně zkompilujeme a teprve po kompilaci a sestavení ho můžeme spustit. U skriptovacího jazyka kompilace není potřebná.

Výhodou skriptovacích programovacích jazyků je, že nemusí být kompilovány po každé úpravě zdrojového kódu a jejich poměrně snadné rozšiřování. Další výhodou může být jejich multiplatformnost, musí pro ně však existovat interpret pro danou platformu.

Nevýhodou může být o něco pomalejší zpracovávání kódu, jelikož je zpracováván při každém spuštění. Další nevýhodou může být větší náročnost na paměť z důvodu nutnosti běhu interpretru.

Typickými zástupci skriptovacích programovacích jazyků jsou například: Perl, Python, PHP (Hypertext Preprocessor), Ruby atd. Také shell z unixových systémů můžeme zařadit do této kategorie.

2.1 Bash

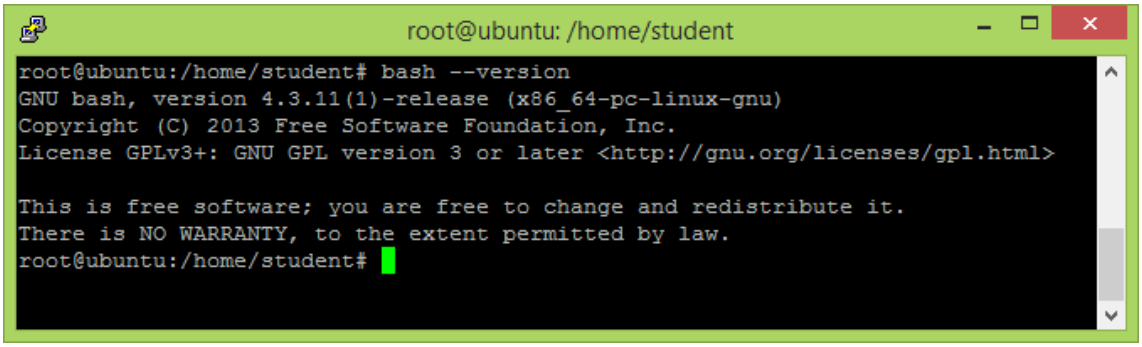
Každý OS (Operating System) pro počítače má nějaký druh uživatelského rozhraní, pomocí kterého může uživatel zadávat příkazy, které má počítač zpracovávat. V Unixových operačních systémech byla vytvořena oddělená součást systému, která umožňuje zadávat příkazy pro OS. Tato součást se nazývá shell. Shellů existuje celá řada a jedním z hojně používaných je právě Bash (Bourne Again shell), zobrazený na obrázku 2.1.

Bash byl vytvořen Brianem Foxem, který jej vytvořil v roce 1989 jako bezplatnou náhradu Bourne shellu. Hlavním úkolem jakéhokoliv shellu je umožnit uživateli ovládat počítač. Shell lze tedy chápat jako prostředníka mezi uživatelem a OS počítače. Když chceme pomocí shellu spustit nějaký program, napíšeme příkaz, kterému bude rozumět, a on za nás obstará spuštění programu. Vše probíhá v textovém režimu.

Jestliže chceme provést celou řadu operací, dostaneme se k vytváření skriptů. Skripty pro shell nejsou nic jiného, než posloupnost příkazů, které má za nás zpracovat. Mnohdy je jednodušší si napsat jednoduchý skript než vypisovat celou řadu příkazů. Více informací o Bashi nalezneme například v [2], [3].

2.2 Perl

Perl patří do rodiny skriptovacích programovacích jazyků šířených pod svobodnou licencí. Byl vytvořen v roce 1987 američanem Larrym Wallem, který chtěl zjednodušit zpracování reportů v Unixových systémech a tak vznikl Perl. Název Perl původně nebyl akronymem, avšak po čase byla komunitou vytvořena různá sousloví, která vystihovala



Obrázek 2.1: Bash s informací o jeho verzi

podstatu Perlu. Jedno z hojně používaných sousloví je: „Practical Extraction and Reporting Language“, což odpovídá původní představě Larryho Walla. Od roku 1987 prošel Perl mnoha změnami a inovacemi. Od prvních verzí už uplynulo mnoho let vývoje a Perl se nyní nachází ve verzi 6. Na jeho vývoj stále dohlíží jeho zakladatel Larry Wall, který je také jedním ze spoluautorů knih, sloužících jako příručky k Perlu [4], [5]. Perl využívá různé prvky z jiných programovacích jazyků a poskytuje především výkonné zpracování textu, bez omezení jejich délky, jak tomu je u jiných shellů. Perl má jinak široké využití, je používán pro programování grafických modelů, správu systémů a mnoho dalšího. Také si umí poradit s funkcionálním, procedurálním a objektovým programováním a může využívat tisíce volně dostupných modulů třetích stran. Více o Perlu se můžete dočíst ve výše zmíněných publikacích nebo využít další dostupné publikace [6], [7].

2.3 Python

Python patří spolu s Perlem k hojně používaným skriptovacím programovacím jazykům. Jeho autorem je holanďan Guido van Rossum, který začal s jeho implementací v roce 1989. Zajímavé jsou okolnosti jeho vzniku. Guido si hledal programátorský projekt, kterým by se přes volno o vánočních svátcích zabýval. V tu dobu se rozhodl, že napíše interpret pro nový programovací jazyk, o kterém již dříve uvažoval. Dal mu pracovní název Python, který mu zůstal i nadále. Guido dodnes dohlíží na vývoj Pythonu. Python je vyvíjen pod svobodnou licencí a je multiplatformní. Aktuálně je Python dostupný ve stabilní verzi 3.4.3 a na jeho vývoji se neustále pracuje.

Zdrojový kód napsaný v Pythonu je oproti jiným jazykům většinou kratší a dobře čitelný. Python, tak jako všechny ostatní programovací jazyky, má svou syntaxi. Ta je zajímavá především tím, že oproti jiným jazykům používá pro ohraničení bloků místo závorek odsazování mezerami či tabulátory.

Tak jako Perl i Python se umí vypořádat s objektovým, funkcionálním i procedurálním programováním a také podporuje využití mnoha modulů. Pro Python není problém spolupracovat s jinými programovacími jazyky. Je původně napsán v jazyku C, existuje i implementace pro Javu (Jython) a pro jazyky, využívající platformnu .NET (IronPython).

Python je využíván v mnoha aplikacích, ať už se jedná o nástroj pro 3D modelování Blender nebo třeba kancelářský balík OpenOffice.org.

Pro více informací můžeme navštívit domovské stránky nebo použít publikace [8], [9].

2.4 Cron

Jelikož je třeba plnit databázi v předem definovaných intervalech, je v podstatě nemožné, aby každé přidání do databáze spouštěl sám uživatel. Abychom mohli opakovaně spouštět námi vytvořený skript v pevně zadaných časových intervalech, bylo zapotřebí nainstalovat SW démona Cron. Instalaci a zprovoznění SW démona Cron se věnuji v následující podkapitole.

2.4.1 Instalace a zprovoznění SW démona Cron

Samotná instalace není složitá, jelikož Crona lze nainstalovat pomocí balíčkovacího systému APT (Advanced Packaging Tool), dostupného v Linuxové distribuci Ubuntu. Provedl jsem to tedy následujícím způsobem:

`$ sudo su` - získání práv uživatele ROOT

`# apt-get update` - aktualizace databáze balíčků

`# apt-get install cron` - samotná instalace Crona

Cron umožňuje vytvořit vlastní soubor pro spouštění daných skriptů pro každého uživatele zvlášť. Pro mou práci jsem využíval účet bez omezených práv - účet ROOT, ve kterém jsem si pomocí Crona vytvořil seznam všech skriptů, které jsem chtěl v daných intervalech spouštět.

Při prvotním spuštění se nás Cron zeptá, jaký editor chceme používat. Já jsem si zvolil NANO, jelikož jsem se s ním už dříve setkal, a práce s ním mi nedělala problémy. Toto nastavení lze i později změnit. Nyní je Cron připraven k použití. Práce s ním je velice jednoduchá. Nejprve ale musíme skriptu, který chceme spouštět, přidělit práva tak, aby byl spustitelný. Za předpokladu, že jsme v adresáři, kde je umístěn skript, to provedeme příkazem:

`# chmod a+x test.pl` nebo alternativním:

`# chmod 755 test.pl`

kde `test.pl` je skript, který chceme automaticky spouštět.

Více o přidělování práv souborům se můžeme dočíst v oficiální dokumentaci k OS Ubuntu [10] nebo přímo v manuálových stránkách příkazu `chmod`, ty spustíme pomocí příkazu: `# man chmod`.

Následující řádky nám ukazují, jak spouštět skript `test.pl` každou minutu:

`# crontab -e` - spuštění Crona

```
* * * * * /home/david/Dokumenty/dp/test.pl
```

Přidáme výše uvedený řádek s cestou ke spouštěnému skriptu.

Editovaný soubor uložíme a opustíme. Při použití editoru NANO jsem použil klávesové zkratky „Ctrl-O“ a „Ctrl-X“. V případě, že bychom si chtěli zkontrolovat, jaké skripty nám Cron automaticky spouští, můžeme použít příkaz:

```
# crontab -l
```

Pro smazání celé tabulky spouštěných skriptů vytvořené pomocí Crona lze použít příkaz:

```
# crontab -r
```

Nyní je skript `test.pl` spouštěn každou minutu. Jestliže chceme automatické spouštění skriptu ukončit, postupujeme stejným způsobem jako při vytváření automatického spuštění, ale pouze s tím rozdílem, že řádek s cestou k danému skriptu umažeme nebo zakomentujeme znakem `#` (to v případě, že předpokládáme budoucí práci se stejným skriptem). Následně editovaný soubor uložíme a opustíme.

A jak jsme tedy docílili, že se bude skript `test.pl` spouštět každou minutu? Onu magii můžeme hledat právě v pěti hvězdičkách, umístěných před cestou ke skriptu. Právě tyto hvězdičky určují s jakým intervalem má být daný skript nebo skripty spouštěny. První hvězdička znamená místo pro minuty (0-59). Když ji zde ponecháme, bude se skript spouštět každou minutu v závislosti na hvězdičkách dalších. Druhá hvězdička zastupuje hodiny (0-23, kdy 0 znamená půlnoc), třetí dny (1-31), čtvrtá měsíce (1-12) a pátá den v týdnu (0-6, kdy 0 je sobota).

Uvedeme si ještě jeden příklad. Chceme, aby se námi vytvořený skript `test.pl` spouštěl každou minutu během běžného pracovního týdne, tj. pondělí až pátek, 6:00 až 14:00 po celý rok. Pro každou minutu z hodiny volíme *, pro hodiny pak 06-13, pro dny volíme *, taktéž pro měsíce, pro dny v týdnu volíme 2-6. Samotný řádek vkládaný do souboru spouštěných skriptů bude vypadat následovně:

```
* 06-13 * * 2-6 /home/david/Dokumenty/dp/test.pl
```

Vše výše uvedené mi bude stačit pro použití při sběru dat, jelikož ve většině případů budou skripty spouštěny každou minutu po blíže neurčenou dobu. Detailnější informace pro použití SW démona Cron nalezneme přímo v manuálových stránkách v systému, které můžeme spustit pomocí příkazu `# man crontab`. Dalším dobrým zdrojem informací je oficiální dokumentace na webových stránkách OS Ubuntu [11].

2.5 Lm-sensors

V OS Ubuntu máme možnost lokálně monitorovat jakoukoliv informaci o HW, kterou jsme schopni dostat ze systému pomocí nějakého příkazu. Ať už se jedná o počet procesů, které získáme pomocí příkazu `ps` nebo o vytížení paměti RAM (Random Access Memory) a příkazu `free`. Některé informace nejsme schopni obstarat pouze pomocí použití daného příkazu a musíme použít nějakého prostředníka, který nám zprostředkuje získávání dalších informací. Tím prostředníkem může být právě nástroj Lm-sensors (Linux monitoring sensors). Lm-sensors nám dává možnost zobrazovat informace například o:

- napětích
  - na jednotlivých větvích zdroje
  - na procesoru
  - na pamětech RAM
  - na čipové sadě
- teplotách
  - na procesoru a jeho jádrech
  - na grafických čipech
  - na základní desce
  - v počítačové skříni
- rychlosti otáček ventilátorů
  - na procesoru
  - na zdroji
  - v počítačové skříni

Zobrazování těchto informací je však závislé na HW konfiguraci počítače nebo notebooku a na existenci ovladačů pro daný HW. Na mém PC mi Lm-sensors poskytl všechny informace, uvedené níže (viz. výpis 1), avšak na mém notebooku jen informace ohledně teplot, na školním virtuálním serveru mi neposkytl informace žádné. Nástroj Lm-sensors je vydáván partou nadšenců pod Open Source licencí. Na jeho vývoji se může podílet každý, ať už návrhem na rozšíření podpory, testováním funkčnosti na daném HW či dalšími připomínkami. Jeho využití je široké, od zvědavců, kteří si chtějí vyzkoušet, co umí, až po zkušené síťové administrátory a overclockery, kteří mají potřebu monitorovat jednotlivé veličiny pro stabilní a nepřerušovaný běh jejich serverů respektive počítačů. Další informace o Lm-sensors můžeme najít na domovských stránkách tohoto projektu [12].

```
root@ubuntu:/home/ubuntu/# sensors
radeon-pci-0100
Adapter: PCI adapter
temp1:        +58.5 C

atk0110-acpi-0
Adapter: ACPI interface
Vcore Voltage:          +1.14 V    (min =  +0.85 V, max = +1.60 V)
 +3.3 Voltage:          +3.30 V    (min =  +2.97 V, max = +3.63 V)
 +5 Voltage:            +4.97 V    (min =  +4.50 V, max = +5.50 V)
 +12 Voltage:           +12.04 V   (min = +10.20 V, max = +13.80 V)
CPU FAN Speed:          2149 RPM   (min =  600 RPM, max = 7200 RPM)
CHASSIS FAN Speed:      1163 RPM   (min =  600 RPM, max = 7200 RPM)
POWER FAN Speed:        1318 RPM   (min =  600 RPM, max = 7200 RPM)
CPU Temperature:        +28.0 C    (high = +60.0 C,  crit = +95.0 C)
```

```
MB Temperature:         +41.0 C       (high = +45.0 C,  crit  = +95.0 C)

k8temp-pci-00c3
Adapter: PCI adapter
Core0 Temp:  +23.0 C
Core0 Temp:  +27.0 C
Core1 Temp:  +28.0 C
Core1 Temp:  +16.0 C
```

Výpis 1: Seznam všech informací poskytnutých senzory

2.5.1 Instalace a zprovoznění nástroje Lm-sensors

Abych mohl nástroj Lm-sensors používat, bylo zapotřebí jej nainstalovat a následně s jeho pomocí nadetekovat všechen podporovaný HW. To jsem provedl následujícími příkazy:

`$ sudo su` - získání práv uživatele ROOT

`# apt-get update` - aktualizace databáze balíčků

`# apt-get install lm-sensors` - samotná instalace Lm-sensors

`# sensors-detect` - detekce podporovaného HW

Po výše provedené automatické detekci daného HW se můžeme pustit do zobrazení informací, poskytnutých pomocí nástroje Lm-sensors. To provedeme následujícím příkazem:

```
# sensors
```

V ideálním případě by nám měl systém po zadání výše uvedeného příkazu vypsat všechny informace, které nám může poskytnout. Celý výpis by měl vypadat tak, jak je uvedeno ve výpisu 1. V případech, kdy nemáme tolik podporovaný HW počítače, bude v tomto výpisu podstatně méně informací. Tolik k základnímu použití nástroje Lm-sensors. Pro širší použití nebo více informací o nástroji Lm-sensors můžeme navštívit buďto domovské stránky [12] nebo si spustit přímo manuálové stránky pomocí příkazu `# man sensors`.

3 RRDtool

RRDtool (Round Robin Database Tool) je nástroj s otevřeným zdrojovým kódem (Open-Source), který slouží pro snadný sběr systémových informací (logů), jejich následné ukládání do databází a vytváření grafů.

Tento nástroj byl založen Tobiasem Oetikerem. Tobias Oetiker je švýcarské národnosti, narodil se 24.února 1969, studoval na Švýcarském federálním technologickém institutu v Curychu, je ženatý a má rád fotografování, umění, cestování a turistiku. V roce 2006 založil společnost OETIKER+PARTNER AG, ve které také pracuje. Je autorem mnoha nástrojů a jedním z nich je právě RRDtool.

RRDtool je napsán v programovacím jayzce C, jeho první verze se datuje na rok 1999 a nyní (2015) je ve verzi 1.5. Jak můžeme na první pohled z názvu poznat, RRDtool využívá pro ukládání dat databází typu Round Robin. Nejedná se o nic jiného, než o databázi s konstantní velikostí, která je definována při jejím vytváření. Je-li například taková databáze definována pro sběr informací po každé minutě během jedné hodiny, tak se v okamžiku, kdy sběr bude pokračovat druhou hodinou databáze nezvětší, ale nově pořízené informace budou přepisovat ty první zapsané - velikost databáze se tím nemění. RRDtool je ve většině případů ovládán skripty. Tyto skripty mohou být napsány v integrovaném interpretru jako je například Bash nebo i jiném skriptovacím programovacím jazyku - Perlu, Pythonu, Ruby atd.

Práce s nástrojem RRDtool se tedy dělí na tři hlavní části:

- Vytvoření databáze - definice dat (pomocí funkce CREATE).
- Sběr dat a většinou cyklické plnění databáze (pomocí funkce UPDATE).
- Vytváření výstupů z databáze - tvorba grafů (pomocí funkce GRAPH).

Jednotlivým částem procesu použití nástroje RRDtool se budeme věnovat v následujících podkapitolách. Popis jednotlivých parametrů funkcí nástroje RRDtool nalezneme v příloze A. Více informací o nástroji RRDtool můžeme nalézt na internetových stránkách projektu [13]. Další informace o autorovi a jeho společnosti nalezneme na [14], [15].

3.1 Instalace RRDtool

Předpokladem pro další kroky je nainstalovaný a zaktualizovaný OS založený na Linuxu. Já jsem si pro monitorování zvolil známou distribuci Ubuntu, konkrétně ve verzi 13.10 pro 64-bitové procesory. Nyní se tedy můžeme pustit do samotné instalace nástroje RRDtool. Tu provedeme pomocí následujících příkazů:

`$ sudo su` - získání práv uživatele ROOT

`# apt-get update` - aktualizace databáze balíčků

`# apt-get install rrdtool` - samotná instalace nástroje RRDtool

`# apt-get install librrds-perl` - instalace knihoven Perlu pro RRDtool

Tímto jsme nainstalovali nástroj RRDtool a nyní se můžeme pustit do vytváření skriptů, které nám budou obstarávat vytváření a plnění databází a následnou tvorbu grafů.

3.2 Funkce CREATE

Funkce CREATE slouží v nástroji RRDtool k vytvoření nové databáze RRD (Round Robin Database), do které se budou následně pomocí funkce UPDATE vkládat data. Tato funkce je definována předpisem (výpis 2) a má několik povinných parametrů, které jsou detailněji popsány v příloze A.1. Takto vytvořená databáze má ihned po vytvoření neměnnou velikost, na pozicích pro budoucí hodnoty jsou prozatímně umístěny hodnoty NaN (Not a Number).

```
rrdtool create filename
[--start|-b start time]
[--step|-s step]
[--no-overwrite]
[DS:ds-name:DST:dst arguments]
[RRA:CF:cf arguments]
```

Výpis 2: Předpis funkce CREATE

3.2.1 Ukázková funkce CREATE

Nyní si ukážeme praktický příklad pro vytvoření databáze RRD. Chceme si monitorovat počet všech spuštěných procesů na serveru. Pro vytvoření databáze použijeme následující skript (výpis 3):

```
rrdtool create pocetProcesu.rrd --step 60 \
DS:pocetProcesu:GAUGE:120:0:500 \
RRA:AVERAGE:0.5:1:1500 \
RRA:AVERAGE:0.5:5:2028 \
RRA:MIN:0.5:5:2028 \
RRA:MAX:0.5:5:2028
```

Výpis 3: Ukázkový skript pro vytváření databáze

A co jsme ve skriptu vlastně provedli? Vezměme si to pěkně popořádku. Vytvořili jsme si databázi RRD, která má název `pocetProcesu.rrd` a očekává informaci s hodnotou počtu spuštěných procesů každých `60` sekund. Jestliže tuto informaci nedostane do `120`ti sekund, zapíše si neznámou (`UNKNOWN`) hodnotu. Také počítá s tím, že bude dostávat informaci z intervalu $\langle 0; 500 \rangle$. Dále zde máme definici několika archivů RRA.

První z nich sbírá aktuální hodnoty pro denní graf - sbírá každou minutu jednu hodnotu po dobu 24 hodin a počítá s hodinovou rezervou (60*24=1440+60=1500 hodnot).

Druhý sbírá aktuální hodnoty pro týdenní graf - zde si získává jednu hodnotu za 5 minut, kterou získá zprůměrováním pěti minutových hodnot. Tyto hodnoty sbírá po celý týden a počítá s hodinovou rezervou (12*24*7=2016+12=2028 hodnot).

Třetí a čtvrtý RRA je definován stejně jako druhý s tím rozdílem, že neukládá aktuální hodnoty, ale hodnoty minimální respektive maximální.

3.3 Funkce UPDATE

Funkce UPDATE nám v nástroji RRDtool slouží pro vkládání dat do předem vytvořené databáze RRD. Tu jsme si v předchozí podkapitole vytvořili pomocí funkce CREATE. Obdobně, jak je tomu u funkce CREATE, tak i funkce UPDATE, má svůj předpis (výpis 4) a několik parametrů, které jsou podrobněji popsány v příloze A.2.

```
rrdtool update filename
[--template|-t ds-name[:ds-name]...]
[--daemon address] [--]
N|timestamp:value[:value ...]
```

Výpis 4: Předpis funkce UPDATE

3.3.1 Ukázková funkce UPDATE

Zde si ukážeme příklad, jak databázi, kterou jsem si vytvořili v podkapitole 3.2.1, naplníme hodnotami. Hodnoty nám budou interpretovat počet spuštěných procesů na serveru. Pro plnění databáze použijeme následující příkaz (výpis 5):

```
rrdtool update pocetProcesu.rrd -- -5:100 N:125
```

Výpis 5: Ukázkový skript pro plnění databáze

Tento příkaz nám vložil do databáze `pocetProcesu.rrd` dvě hodnoty. První s hodnotou `100` a časovou značkou, odpovídající času před pěti sekundami. Druhý pak nese časovou značku s aktuálním časem a hodnotu `125`.

3.4 Funkce GRAPH

Hlavním úkolem funkce GRAPH, nástroje RRDtool, je zobrazit data, uložená do databáze RRD ve formě čitelné pro člověka. Funkce GRAPH vytváří grafy, reprezentující číselné hodnoty, uložené v databázi. Tak jako předchozí funkce, i funkce GRAPH, má svůj předepsaný tvar (výpis 6) a parametry, které jsou detailně probrány v příloze A.3.

```
rrdtool graph filename
[option ...]
[data definition ...]
[data calculation ...]
[variable definition ...]
[graph element ...]
[print element ...]
```

Výpis 6: Předpis funkce GRAPH

3.4.1 Ukázková funkce GRAPH

V dřívějších ukázkách - u funkce CREATE (podkapitola 3.2.1) a UPDATE (podkapitola 3.3.1) jsme si vytvořil databázi a poté ji naplnili daty. Teď je na řadě to nejdůležitější, proč jsme to všechno dělali, a to je vykreslování grafů. V celé kapitole jsme se věnovali různým možnostem vykreslování grafů s nespočtem vlastních voleb a možností. Proto si zde ukážeme, jak vytvořit defaultní graf (obrázek 3.1) a jak moc ho můžeme vlastní tvorbou pomocí těchto parametrů pozměnit (obrázek 3.2). Obrázek 3.1 a 3.2 nám zobrazuje totožná data, pocházející ze stejné databáze RRD. Pro jejich vytvoření jsme použili skript, umístěný ve výpisu 7, respektive výpisu 8. Oba skripty jsou vytvořeny v jazyce Perl.

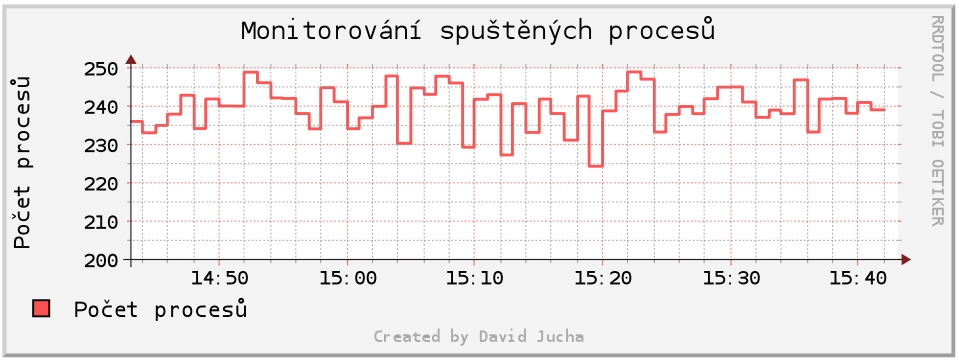
```
\#!/usr/bin/perl
use RRDs;

my $rrd="/home/ubuntu/Documents/rrd/pocetProcesu.rrd";
my $graphDefault="/home/ubuntu/Documents/rrd/pocetProcesuDefault.svg";

RRDs::graph "$graphDefault",
"--start=now-1h",
"--end=now",
"--title=Monitoring spustenych procesu",
"--vertical-label=Pocet procesu",
"--watermark=Created by David Jucha",
"--imgformat=SVG",
"DEF:proc=$rrd:proc:AVERAGE",
"LINE1.5:proc#FF0000AA:Pocet procesu\\n"
;
```

Výpis 7: Ukázkový skript pro vytvoření defaultního grafu

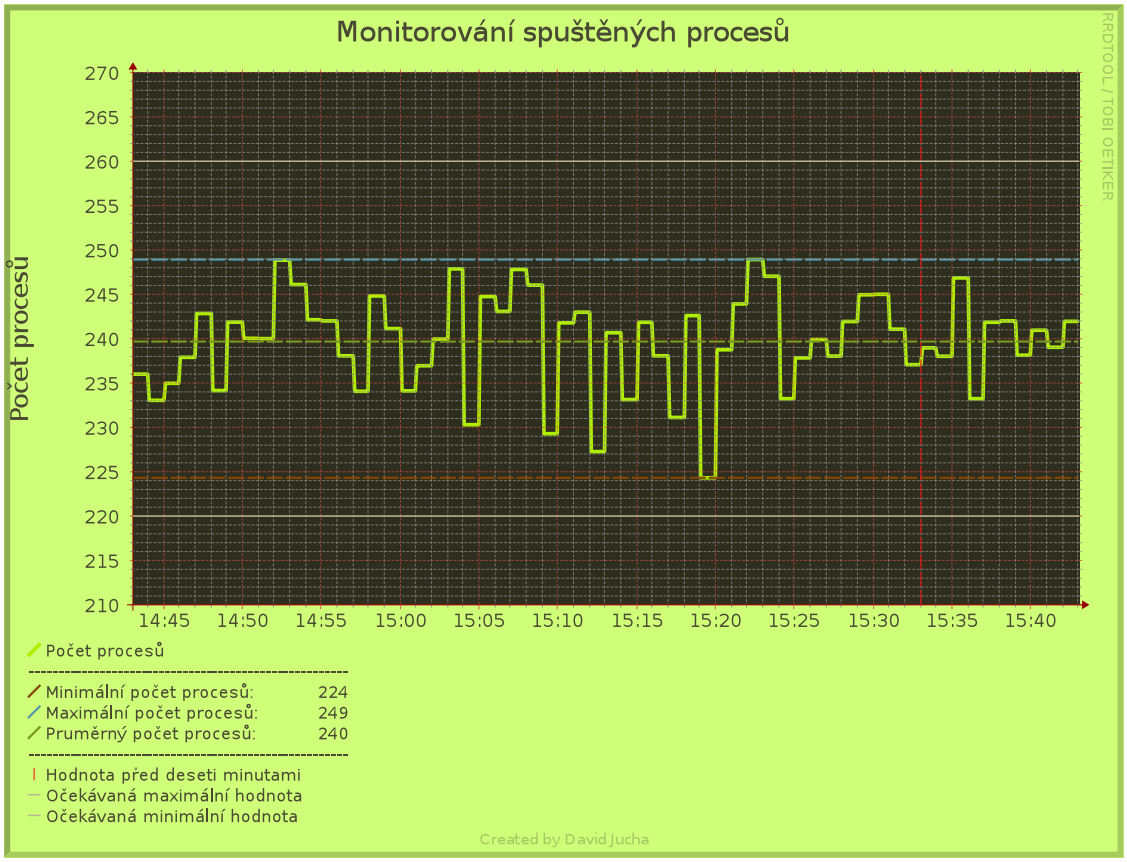
```
#!/usr/bin/perl
use RRDs;

my $vr=time-600;
my $rrd="/home/ubuntu/Documents/rrd/pp/pocetProcesu.rrd";
my $graph="/home/ubuntu/Documents/rrd/pp/pocetProcesu.svg";

RRDs::graph "$graph",
"--start=now-1h",
"--end=now",
"--title=Monitoring spustenych procesu",
"--vertical-label=Pocet procesu",
"--watermark=Created by David Jucha",
"--imgformat=SVG",
### OTHER ###
"--width=640",
"--height=360",
"--rigid",
"--upper-limit=240",
"--lower-limit=180",
"--x-grid=MINUTE:1:MINUTE:5:MINUTE:5:0:%R",
"--grid-dash=2:1",
"--border=4",
```

```
"--dynamic-labels",
"--slope-mode",
### COLORS ###
"--color=BACK#D1FF78",
"--color=CANVAS#2E2E1F",
"--color=SHADEA#92B254",
"--color=SHADEB#A7CC60",
"--color=FONT#4C4C33",
"--color=AXIS#990000",
"--color=ARROW#990000",
### FONTS ###
"--font=DEFAULT:0:ZapfDingbats",
"--font=AXIS:9",
"--font=UNIT:12",
"--font=TITLE:14",
"--font=LEGEND:8",
"--font=WATERMARK:7",
### DATA DEFINITION ###
"DEF:proc=$rrd:proc:AVERAGE",
"VDEF:minProc=proc,MINIMUM",
"VDEF:maxProc=proc,MAXIMUM",
"VDEF:avgProc=proc,AVERAGE",
### LINES AND LEGEND ###
"LINE2.5:proc#AFEE00:␣Pocet␣procesu\\n:",
"COMMENT:----------------------------------------------------------\\n",
"LINE1.5:minProc#854600:␣Minimalni␣pocet␣procesu\\:\\t:dashes=10,3",
"GPRINT:minProc:\%6.0lf\\n",
"LINE1.5:maxProc#5599AA:␣Maximalni␣pocet␣procesu\\:\\t:dashes=10,3",
"GPRINT:maxProc:\%6.0lf\\n",
"LINE1.5:avgProc#789521:␣Prumerny␣pocet␣procesu\\:\\t:dashes=10,3",
"GPRINT:avgProc:\%6.0lf\\n",
"COMMENT:----------------------------------------------------------\\n",
"VRULE:$vr#ff0000AA:␣Hodnota␣pred␣deseti␣minutami\\n:dashes=10,2",
"HRULE:230#B8B894:␣Ocekavana␣maximalni␣hodnota\\n",
"HRULE:190#B8B894:␣Ocekavana␣minimalni␣hodnota\\n"
;
```

Výpis 8: Ukázkový skript pro vytvoření upraveného grafu



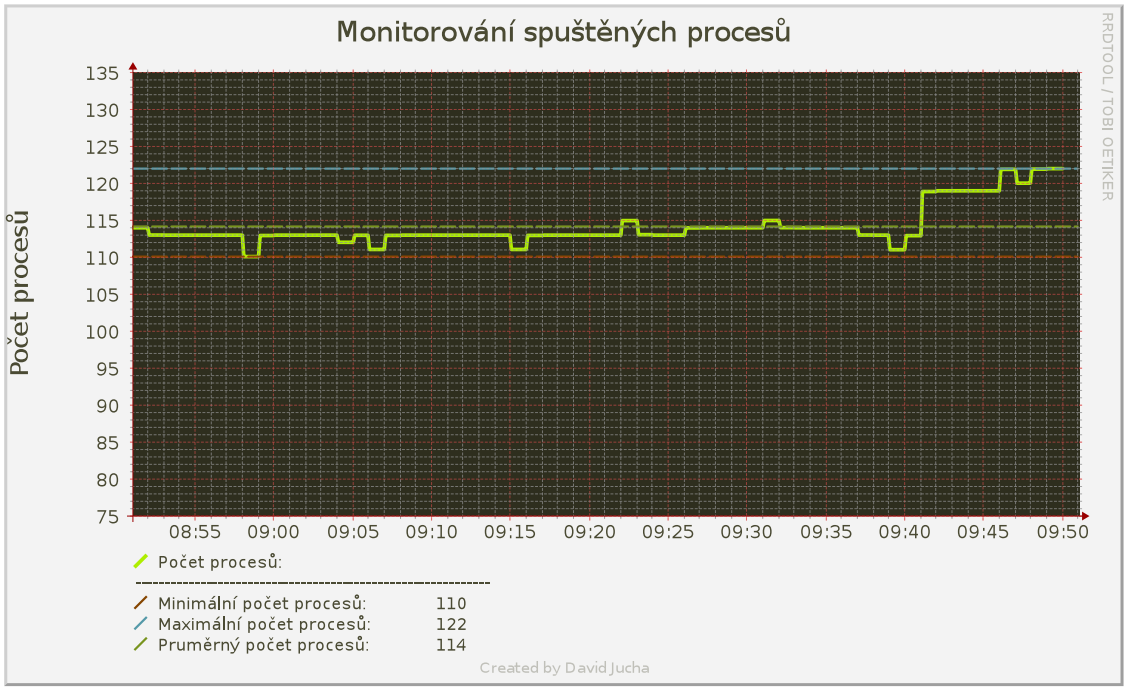
Obrázek 3.1: Defaultně vytvořený graf s minimem parametrů



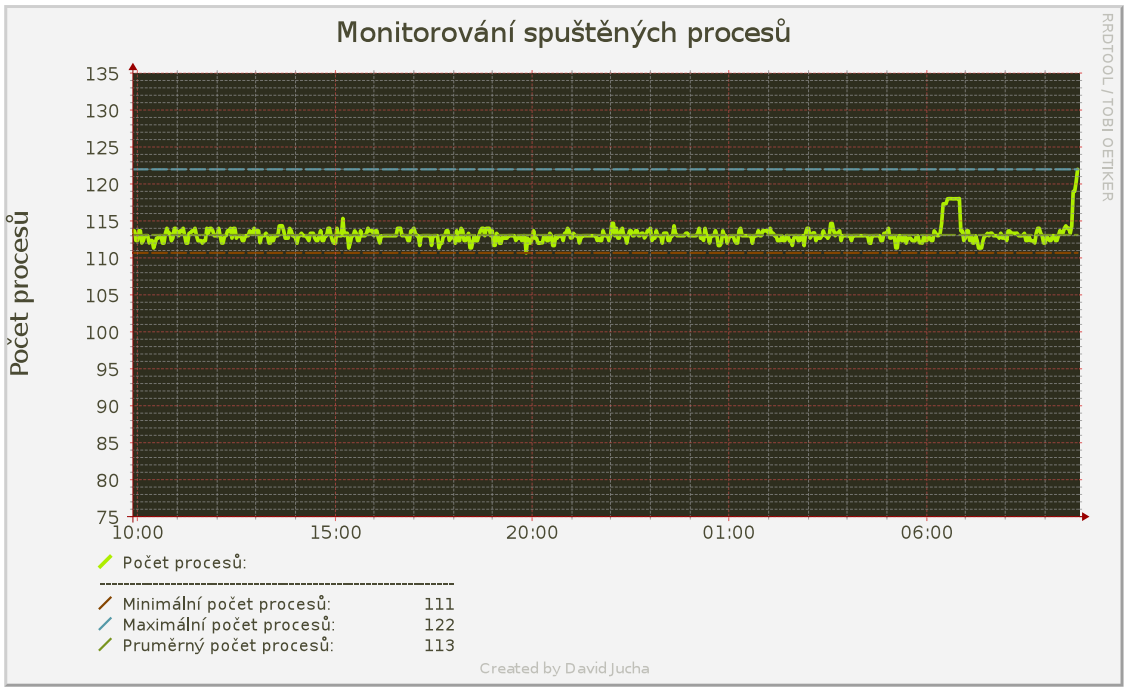
Obrázek 3.2: Upravený graf použitím mnoha parametrů

3.4.2 Ukázková funkce GRAPH zobrazující data za poslední měsíc

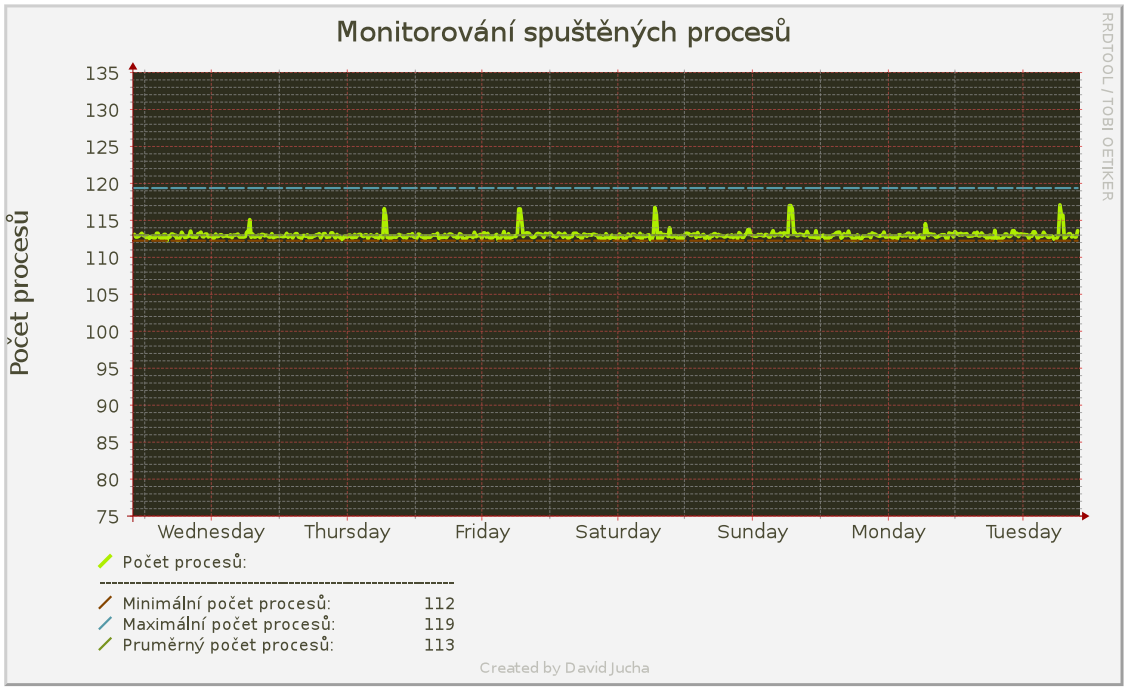
Jak jsme se v dřívějších kapitolách dověděli, nástroj RRDtool umožňuje vytvářet dlouhodobé grafy. RRDtool neomezuje maximální délku vytvářených grafů a je jen otázkou, za jaké maximální časové období má smysl informace sbírat a analyzovat. Pro názornou ukázku jsem nechal běžet na virtuálním serveru, vytvořeném pro účely diplomové práce (DP1) skript, který mi monitoroval počet spuštěných procesů na tomto serveru. Cílem bylo ukázat naměřené hodnoty za poslední hodinu (obrázek 3.3), den (obrázek 3.4), týden (obrázek 3.5) a měsíc (obrázek 3.6), protože u všech ostatních měření je maximální doba sběru dat z časových a jiných důvodů zkrácena na dvanáct hodin. Dlouhodobější grafy využívají například poskytovatelé internetového připojení. Ti z nich následně mohou vyčíst pro ně důležité informace o vytížení jejich jednotlivých síťových uzlů, o špičkách v provozu nebo o stále rostoucích požadavcích na rychlost připojení.



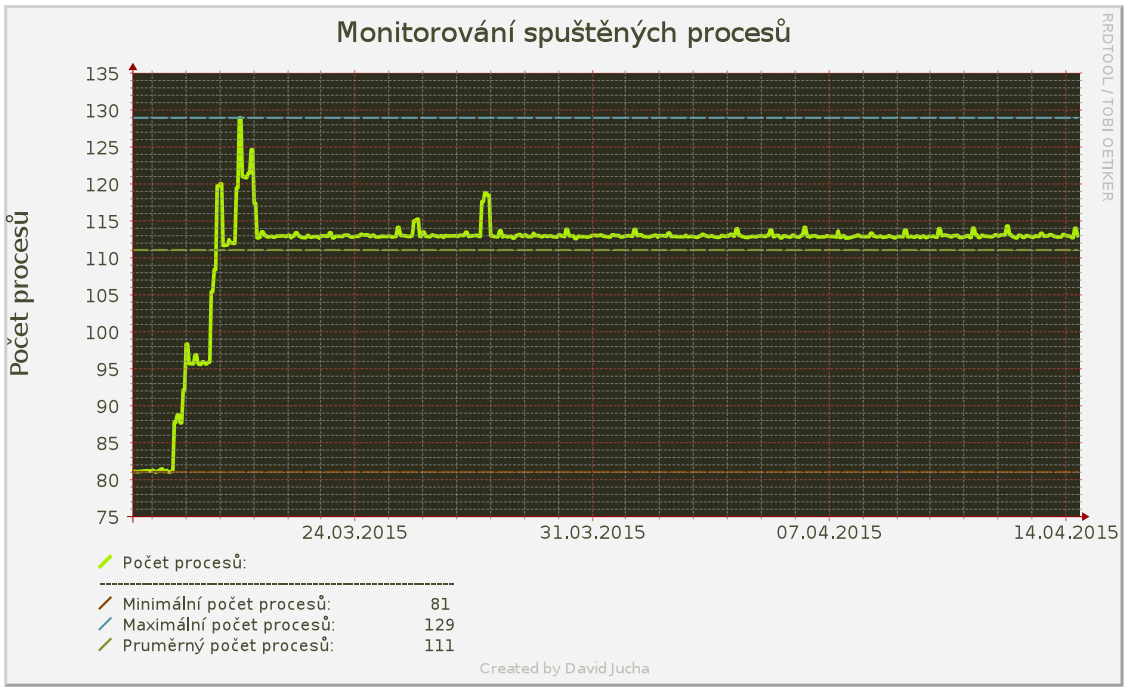
Obrázek 3.3: Počet spuštěných procesů na stanici DP1 za poslední hodinu



Obrázek 3.4: Počet spuštěných procesů na stanici DP1 za poslední den



Obrázek 3.5: Počet spuštěných procesů na stanici DP1 za poslední týden



Obrázek 3.6: Počet spuštěných procesů na stanici DP1 za poslední měsíc

4 Sběr a zpracování systémových a síťových informací pomocí nástroje RRDtool

V následujících podkapitolách se budu zabývat praktickou částí práce - sběrem různých informací v různých situacích a s různými zařízeními. Nejprve se bude jednat o lokální sběr informací ze stanic, následně půjde o sběr informací ze stanic připojených k síti a nakonec půjde o sběr informací z různých jiných zařízení připojených k síti. Stanici můžeme chápat PC, notebook nebo server. V případě jiných zařízení se může jednat o router, switch, NAS či mobilní telefon.

V každém měření ve všech podkapitolách je vždy ukázán jeden graf zobrazující informace za poslední hodinu měření. V přílohách D.1, D.2 a D.3 odpovídajících podkapitolám 4.1, 4.2, 4.3 a přiložených C.1, C.2, C.3 je vždy ukázán jeden z dlouhodobějších grafů. Všechny grafy, skripty a databáze RRD z každého měření jsou umístěny v příloze na CD (Compact Disc) v odpovídajících adresářích. U každého měření je kladen důraz na vysvětlení, jakým způsobem a pomocí jakých příkazů získáváme data pro zobrazování do grafů. V příloze B je umístěn ukázkový skript pro jedno měření napsaný v jazyce Perl, který jsem si zvolil jako hlavní jazyk pro vytváření skriptů, jelikož po vyzkoušení ostatních jazyků mi přišla práce právě s Perlem jako nejlepší volba. V jednom skriptu jsou vždy obsaženy všechny 3 hlavní funkce - Create, Update a Graph.

Součástí práce bylo také vytvořit návody pro praktickou výuku. Byly tedy vytvořeny čtyři stručné návody popisující jednotlivé typy měření, odpovídající kapitolám 4.1 (2 návody), 4.2 a 4.3. Všechny návody jsou umístěny v příloze E a slouží čtenářům k nastínění obecných postupů při požadavcích monitorování síťových nebo systémových informací pomocí nástroje RRDtool.

4.1 Lokální sběr informací z PC, notebooku nebo serveru

V této podkapitole jsem se zaměřil na lokální sběr informací. Pod tímto si ale každý může představit něco jiného, tak to trochu upřesním. Lokálním je myšleno, že sběr a prezentace informací probíhá na stejném zařízení, ze kterého jsou tato data čerpána. Na sběr nemá vliv to, jestli je zařízení připojeno k síti či nikoliv. Není zde využíván protokol SNMP.

Tabulka 4.1: Konfigurace stolního PC

Komponenta	Výrobce	Typ
Základní deska	ASUS	M3A78
Procesor	AMD	Athlon 64 X2 6000+ 3.1 GHz
Zdroj	Fortron	Blue Storm II 500 W (FSP500-GLN)
Grafická karta	ATI	Radeon HD 4670 512 MB
Paměť RAM	OCZ	Platinium XTC (kit 2x 2 GB) 1000 MHz

V měřeních 4.1.1, 4.1.2 a přiložených C.1.1 až C.1.4 je využíváno nástroje Lm-sensors (viz kapitola 2.5), který nám umožňuje získávat informace o napětích, teplotách a otáčkách

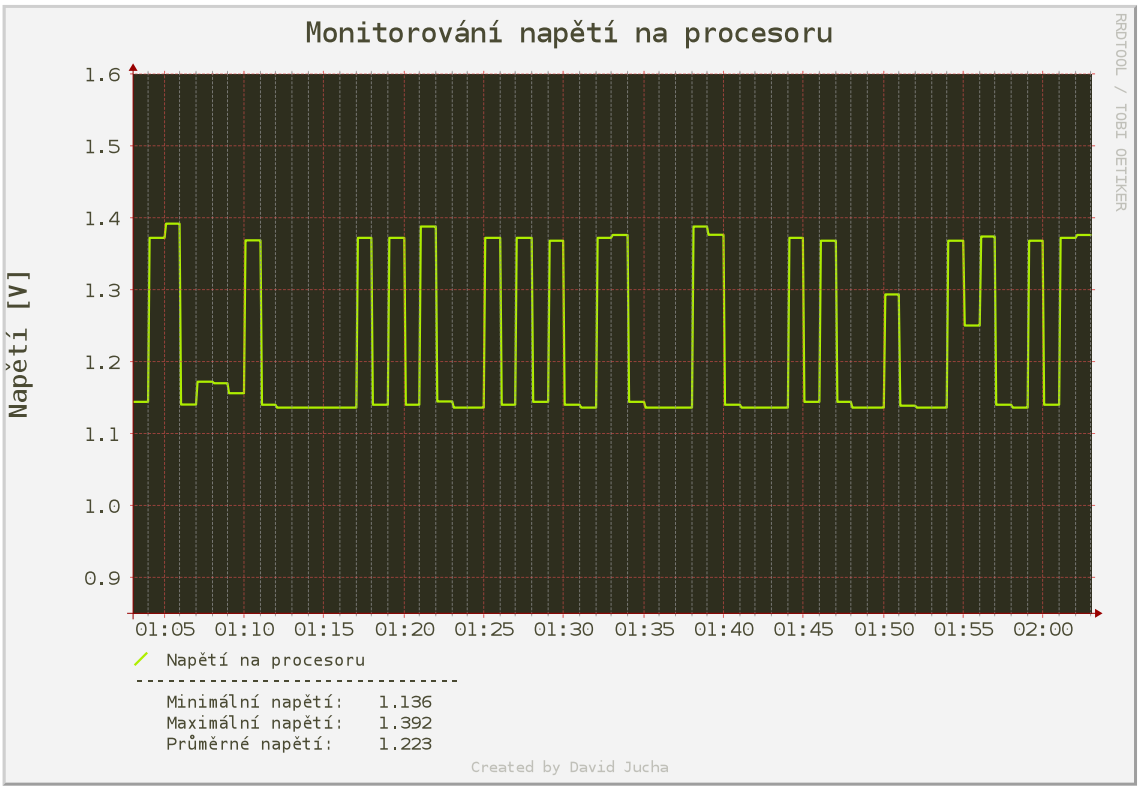
ventilátorů. V ostatních měřeních - tj. měření 4.1.3, 4.1.4 a přiložených C.1.5 až C.1.7 je využíváno pouze nástrojů, integrovaných v OS Ubuntu.

Před samotnými měřeními jsem si ověřil, který počítač, notebook nebo server mi poskytne nejvíce informací pomocí nástroje Lm-sensors. Tím počítačem byl můj domácí stolní PC. Jeho konfigurace, která má vliv na použití nástroje Lm-sensors, je umístěna v tabulce 4.1.

4.1.1 Napětí na procesoru (tzv. vcore napětí)

Jedním z možných monitorování je monitorování napětí na procesoru (obrázek 4.1). To může být velice užitečné například pro overclockery, kteří si chtějí hlídat stabilitu jejich přetaktovaného procesoru nebo pro serverové administrátory, kteří chtějí mít informace o napětí pod dohledem.

Informace o napětích lze získat ze systému pomocí příkazu: `# sensors`. To pro nás ovšem není dostačující, jelikož nepotřebujeme získat celý výpis všech možných napětí, ale jen číselnou hodnotu, zobrazující napětí na procesoru. Pro jednodušší zpracování použijeme přepínač `-u`, který nám umožní jednodušší použití následujícího příkazu, jelikož zobrazí informace ve tvaru: `typ_hodnoty: hodnota`.



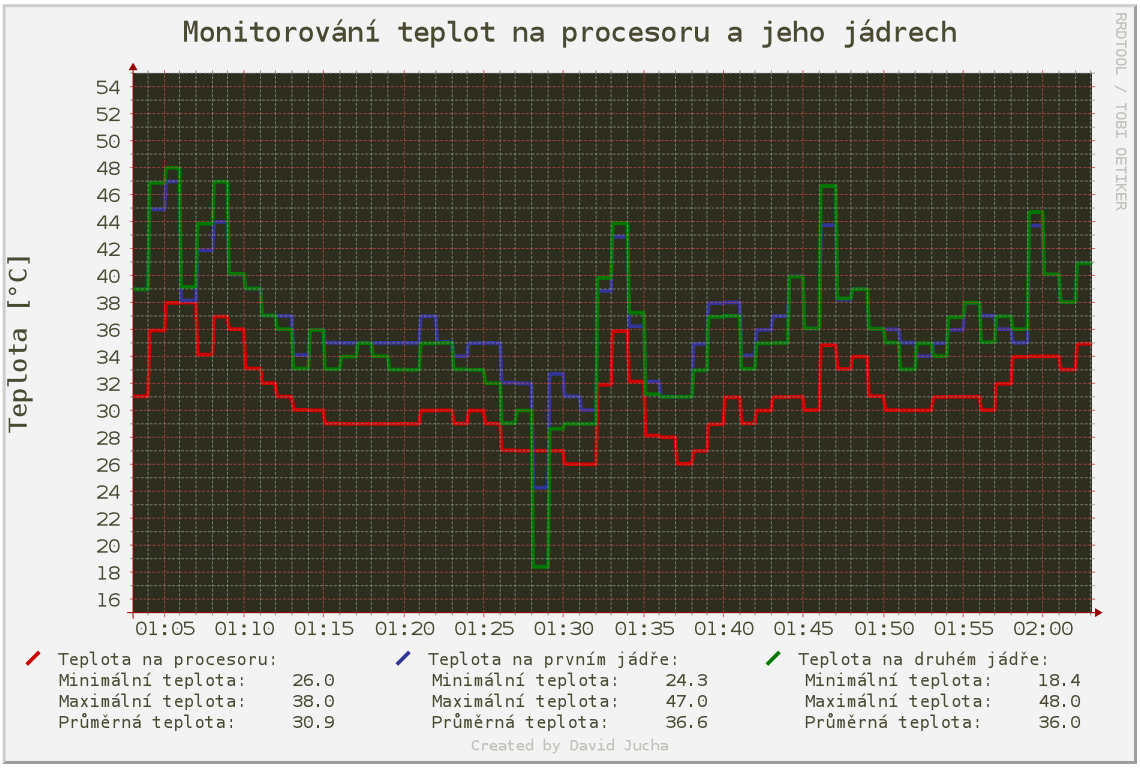
Obrázek 4.1: Napětí na procesoru za poslední hodinu

Nyní si s pomocí příkazu `awk` vybereme `typ_hodnoty`, se kterou chceme zobrazit daný řádek. V našem případě to byl `typ_hodnoty in0_input`. Následně si ještě v příkazu `awk` zvolíme, že chceme zobrazit jen hodnotu bez jejího typu. To provedeme tak, že příkazu `awk` sdělíme, že chceme „vytisknout“ jen druhý sloupec. Oba dva příkazy zřetězíme a dostaneme výsledný příkaz, který bude vypadat následovně:

```
# sensors -u | awk '/in0_input/ {print $2}'
```

4.1.2 Teplota na procesoru a jeho jádrech

Dalším měřením v pořadí je měření teplot. Dle mě je toto měření řádově častěji používané, než předchozí měření. Z naměřených hodnot můžeme vyhodnocovat kvalitu chlazení při různém vytížení procesoru dané stanice. Zajímavé může být sledovat rozdíly mezi teplotami jednotlivých jader při jejich zatížení, či vliv jednoho jádra na teplotu celého procesoru (obrázek 4.2).



Obrázek 4.2: Teplota na procesoru a jeho jádrech za poslední hodinu

Pro získávání informací ohledně teplot procesoru a jeho jader se zprvu postupuje obdobně, jako u výše uvedeného měření. Použití příkazu `# sensors` s parametrem `-u` je obdobné. Změny nastávají v typech hodnot, zde se jedná o řetězce `temp1_input`, `temp1_input` a `temp4_input`. Už z těchto řetězců můžeme vyčíst, že teplota procesoru a prvního jádra má stejný `typ_hodnoty` a že bude nutné je nějak rozlišit. Řetězec

`temp1_input` obsahují ve výpisu nástroje `# sensors` tři řádky a nás zajímá ten druhý pro teplotu procesoru a ten třetí pro teplotu prvního jádra. Jedním z řešení je výstup znovu zřetězit dalším příkazem `awk`, ve kterém si vybereme zobrazení jen určitého řádku. Chceme-li například vybrat ten druhý, druhý příkaz `awk` bude vypadat následovně `awk 'NR == 2'`. Také zde nepotřebujeme teplotu na tři desetinná místa ve tvaru `XX.XXX`, provádíme tedy `substr` s parametry `$2,1,4` a dostáváme hodnotu s jedním desetinným místem ve tvaru `XX.X`.

Výsledné příkazy pro teploty procesoru a jejich jader mají finální tvar:

```
# sensors -u | awk '/temp1_input/ {print substr($2,1,4)}'
| awk 'NR == 2'

# sensors -u | awk '/temp1_input/ {print substr($2,1,4)}'
| awk 'NR == 3'

# sensors -u | awk '/temp4_input/ {print substr($2,1,4)}'
```

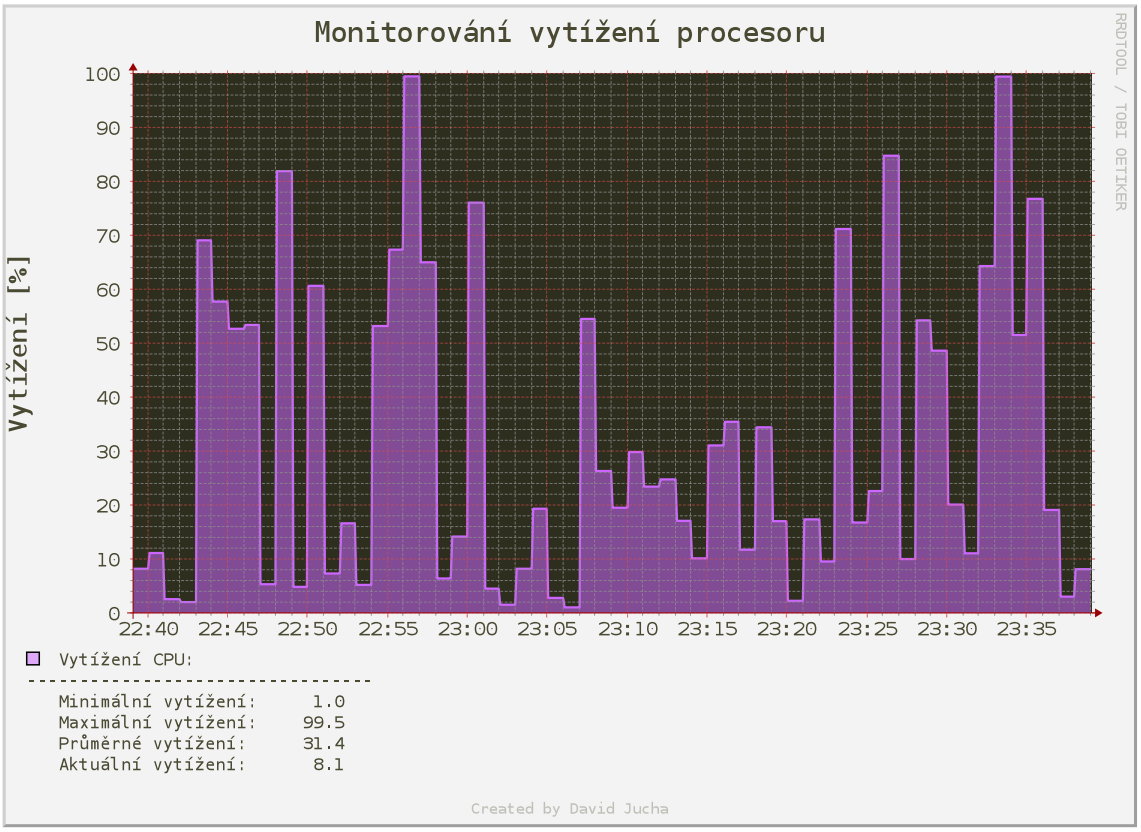
Z naměřeného grafu (obrázek 4.2) můžeme vyčíst závislost teploty jednotlivých jader na celkové teplotě na procesoru a také to, že teplota jader je v převážné části měření vyšší, než teplota na procesoru.

4.1.3 Vytížení procesoru

Pod pojmem vytížení procesoru si můžeme představit, jak moc je procesor stanice využíván v čase. Hodnoty vytížení je nejjednodušší zobrazovat v procentech, jelikož jsou tyto hodnoty i pro člověka nejlépe pochopitelné (obrázek 4.3). Vytížení může síťové administrátory informovat o využití či nevyužití jejich serverů a také je může informovat o nadměrném vytěžování jednotlivých serverů a tím o zkracování jejich životnosti.

K hodnotám o vytížení procesoru se dá dostat několika způsoby, ne všechny jsou však spolehlivé, jelikož volání samotného příkazu ke zjišťování hodnoty může procesor na okamžik poměrně vytížit a tím zkreslit výslednou hodnotu. Z několika možností jsem nakonec vybral za nejspolehlivější tu níže popsanou. Další možností je využít SNMP, ale o tom se zmíním až v následující kapitole, kde budu SNMP využívat pro zjišťování všech informací.

Pro získání informací nejen o vytížení procesoru slouží v OS Ubuntu příkaz `top`. Ten nám poskytuje poměrně veliké množství informací, ale nás bude zajímat pouze třetí řádek obsahující řetězec `Cpu(s)`, který nese informace právě o vytížení procesoru. Příkaz `top` je nutné spouštět ve více iteracích, jelikož ta první nám nedává korektní informace, proto jej voláme ve třech iteracích parametrem `-bn 3`. Čas mezi iteracemi si nastavíme na jednu sekundu pomocí parametru `-d 1`. Následně hledáme pomocí příkazu `awk` výše zmíněný řetězec `Cpu`. Tím omezíme výpis na tři řádky (odpovídající třem iteracím). Každý řádek obsahuje osm hodnot spolu s jejich popisem, nás bude zajímat ta, která je popsána jako `id`, což je zkratka pro `idle`, znamenající procento nečinnosti procesoru. Nyní potřebujeme vybrat třetí řádek a v něm hodnotu `idle`, kterou odečteme od sta a získáme tak aktuální vytížení. Opět použijeme příkaz `awk` s parametrem `NR == 3` a potřebujeme „vytisknout“ hodnotu „100 - hodnota v osmém sloupci“, tj. `print 100-$8`.



Obrázek 4.3: Vytížení procesoru za poslední hodinu

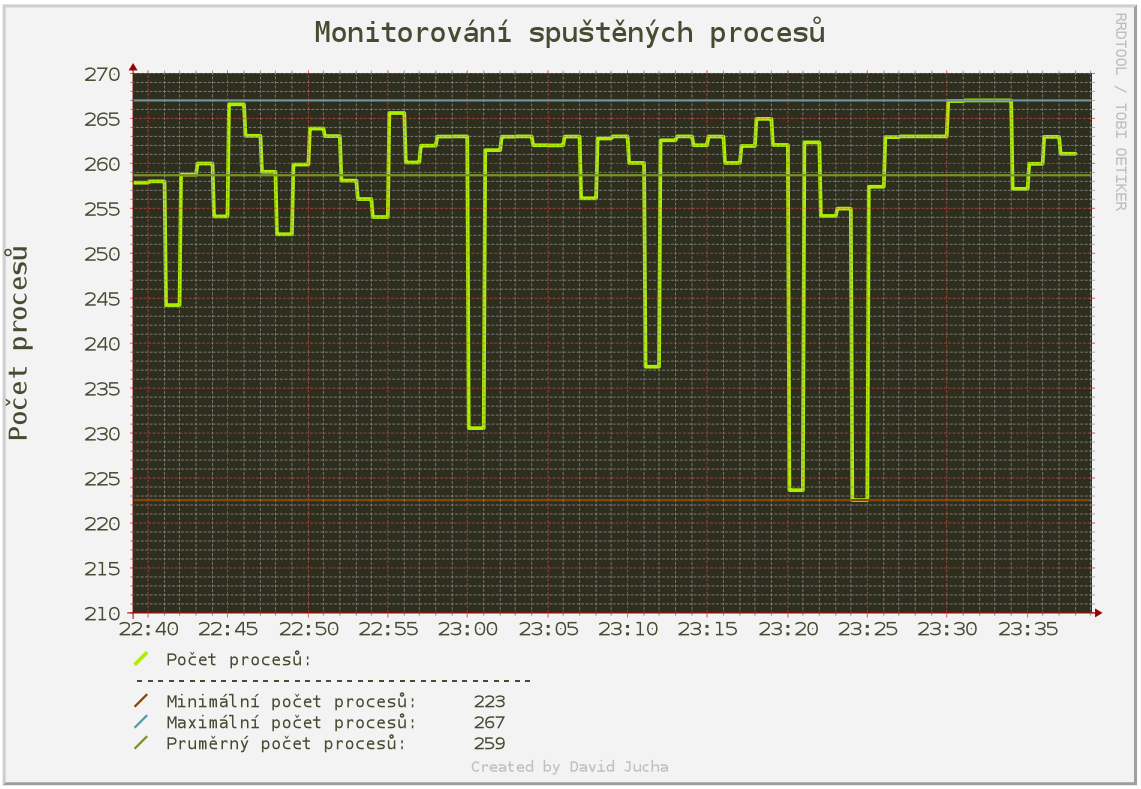
Výsledný příkaz pro získávání hodnoty aktuálního vytížení má následující podobu:

```
# top -bn 3 -d 1 | awk '/Cpu/' | awk 'NR == 3 {print 100-$8}'
```

4.1.4 Počet spuštěných procesů

I počet spuštěných procesů na dané stanici může být zajímavé monitorovat (obrázek 4.4). Můžeme z něj vysledovat, kdy probíhají různé automatické úkony systému anebo kdy jsou naše server nejvíce vytěžovány. Případně můžeme sledovat, jak počet spuštěných procesů ovlivňuje teploty na procesoru.

Pro získání informací o procesech v OS Ubuntu slouží příkaz `ps`. Ten nám poskytuje mnoho informací o spuštěných procesech, jejich PID (process identifier), času běhu nebo příkazu, který daný proces spustil. V případě, že chceme monitorovat celkový počet procesů, běžících na stanici, musíme použít parametr `ax`. Abychom zjistili jejich počet, použijeme ještě příkaz `wc`, který nám umí s parametrem `-l` spočítat řádky, které v našem případě odpovídají počtu všech spuštěných procesů. Abychom byli úplně korektní, je třeba od celkového počtu odečíst jedničku, jelikož je do celkového počtu započtena i hlavička s popisem. Tuto úpravu však provádím až při psaní skriptů v jazyce Perl, takže zde není v příkazu uvedena.



Obrázek 4.4: Počet spuštěných procesů za poslední hodinu

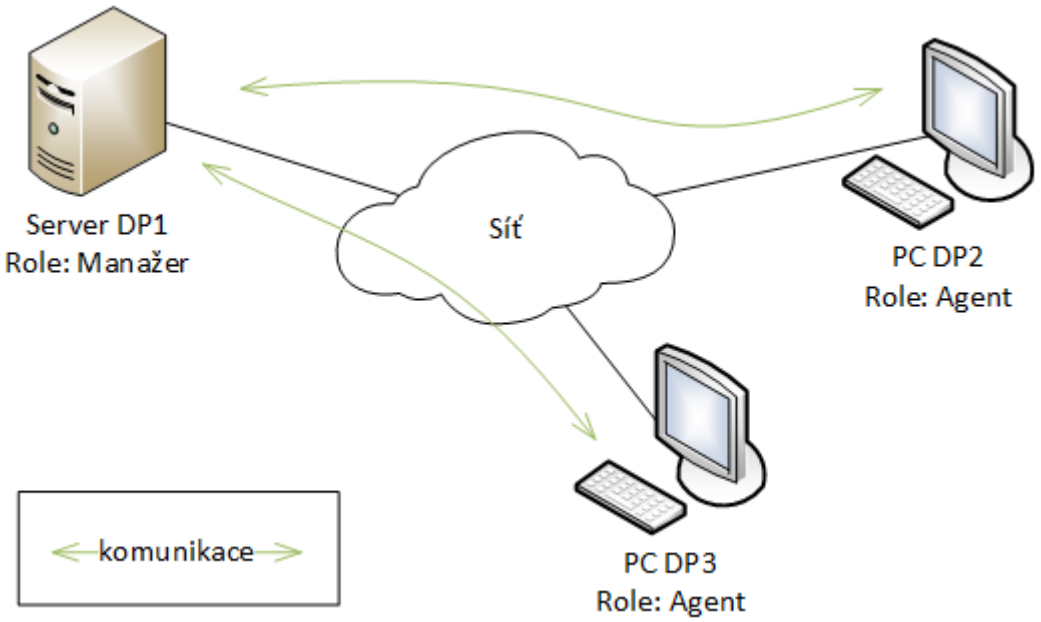
Výsledný příkaz pro zjištění počtu procesů vznikne tedy zřetězením příkazu `ps` spolu s příkazem `wc` a má tvar:

```
# ps ax | wc -l
```

4.2 Sběr informací z PC, notebooků nebo serverů v počítačové síti

Tato podkapitola se zabývá měřeními v počítačové síti. Může se jednat například o firemní síť nebo o síť providera, poskytujícího internet. Při měřeních 4.2.1, 4.2.2 a přiložených C.2.1 až C.2.3 bylo využíváno protokolu SNMP (vyjma posledního přiloženého měření C.2.4).

Vytvořená síť byla zapojena podle schématu, uvedeného na obrázku 4.5. Všechna měření byla zpracovávána virtuálním serverem DP1, který získával data z virtuálních klientských stanic DP2 a DP3. Tyto virtuální servery byly umístěny ve školní serverovně a přistupoval jsem na ně z místa mého bydliště použitím sítě VPN (Virtual Private Network) a nástroje Putty.



Obrázek 4.5: Schéma zapojení sítě, na které byla prováděna měření

4.2.1 Využití místa na pevném disku stanice DP2 a DP3

Pomocí protokolu SNMP můžeme například monitorovat velikost volného místa na disku nebo jeho využití. V jednom grafu (obrázek 4.6) jsou zde zobrazeny hodnoty sesbírané ze dvou stanic. Pro zajímavost a lepší čitelnost jsem jedno obsazení disku umístil nad osu X a druhé obsazení pod osu X. Rozsah grafu na ose Y udává použitelnou velikost disků obou stanic. Disky mají shodnou velikost, proto je osa X zobrazena ve vertikální polovině grafu. Pod grafem se nám zobrazují informace o aktuálním využití místa. Získávané hodnoty se opět nacházejí v databázi `UCD-SNMP-MIB`, která odpovídá podstromu `.1.3.6.1.4.1.2021`.

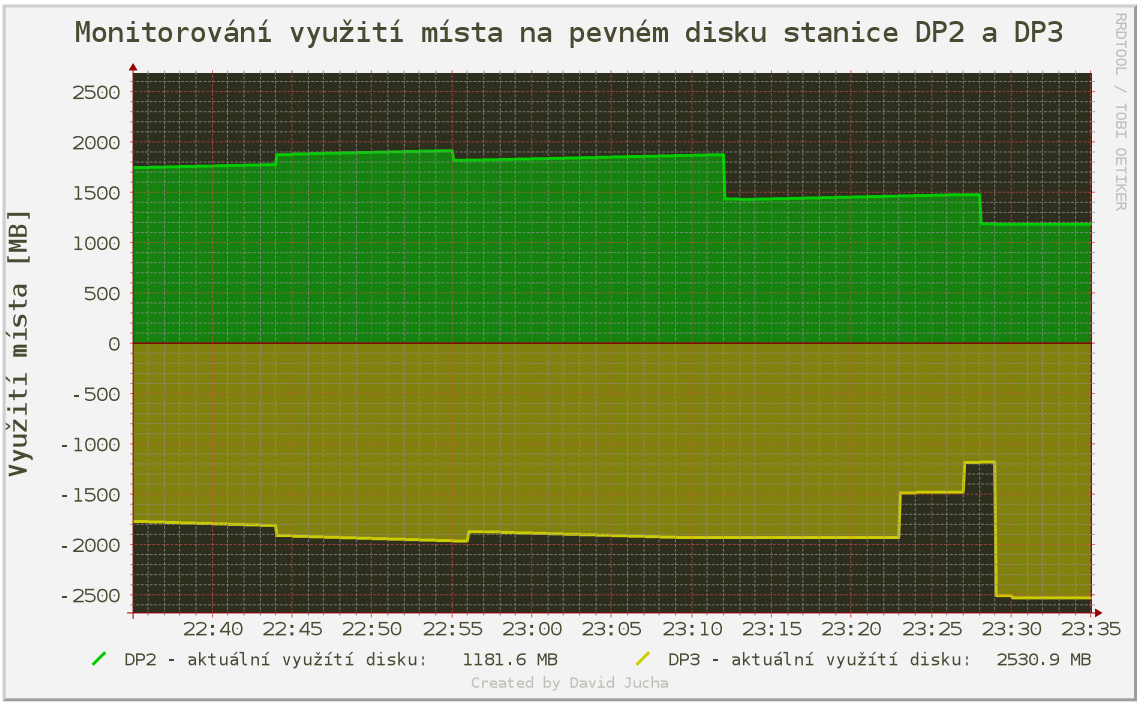
Přesná cesta ve stromové struktuře MIB je následující:

```
.1.3.6.1.4.1.2021.9.1.8.1 - dskUsed.1
```

Celé příkazy pro získávání informací o využití místa na discích mají následující tvar:

```
snmpget -v 2c -c public -m UCD-SNMP-MIB -Oqv
udp6:[2001:718:1001:2c6::182] dskUsed.1
```

```
snmpget -v 2c -c public -m UCD-SNMP-MIB -Oqv
udp6:[2001:718:1001:2c6::183] dskUsed.1
```



Obrázek 4.6: Využití místa na pevném disku stanice DP2 a DP3 za poslední hodinu

4.2.2 Vytížení paměti RAM na stanici DP2 a odděleně i na DP3

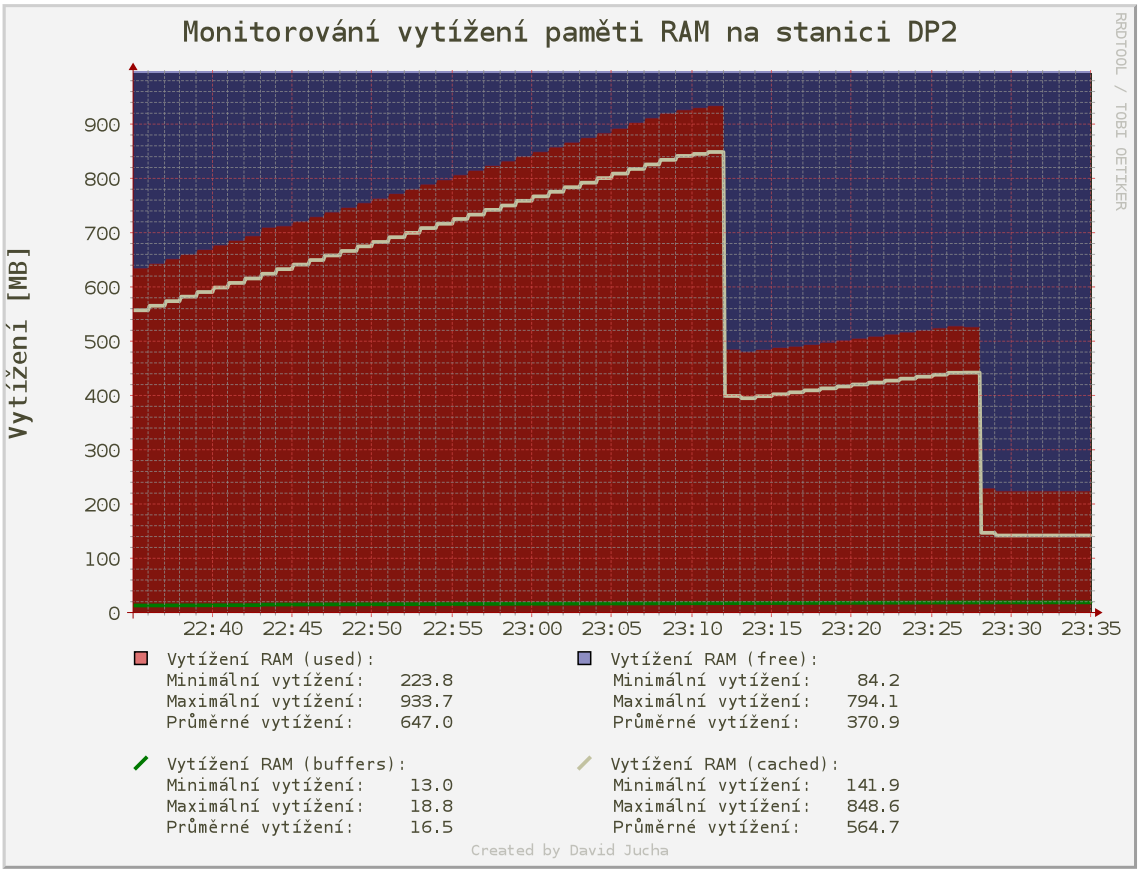
Někdy může být potřeba monitorovat vytížení pamětí RAM na jednotlivých stanicích. Opět nám k tomu s využitím SNMP pomůže databáze `UCD-SNMP-MIB`, která odpovídá podstromu `.1.3.6.1.4.1.2021`. Pro každou stanici zobrazuji v grafu (obrázek 4.7) více informací o RAM paměti. Konkrétně se jedná o velikosti bufferů, cache, využitého a volného místa. Grafy pro stanici DP3 jsou umístěny v příloze na CD. Celá cesta ve stromové struktuře databáze MIB je následující:

```
.1.3.6.1.4.1.2021.4.5 - memTotalReal.0
.1.3.6.1.4.1.2021.4.6 - memAvailReal.0
.1.3.6.1.4.1.2021.4.14 - memBuffer.0
.1.3.6.1.4.1.2021.4.15 - memCached.0
```

Celé příkazy pro získávání informací o pamětech RAM a jejich využití pro stanici DP2 jsou zobrazeny níže. Pro stanici DP3 je potřeba změnit IP adresu.

```
snmpget -v 2c -c public -m UCD-SNMP-MIB -Oqv
udp6:[2001:718:1001:2c6::182] memTotalReal.0

snmpget -v 2c -c public -m UCD-SNMP-MIB -Oqv
udp6:[2001:718:1001:2c6::182] memAvailReal.0

snmpget -v 2c -c public -m UCD-SNMP-MIB -Oqv
udp6:[2001:718:1001:2c6::182] memBuffer.0
```

```
snmpget -v 2c -c public -m UCD-SNMP-MIB -Oqv
udp6:[2001:718:1001:2c6::182] memCached.0
```



Obrázek 4.7: Vytížení paměti RAM na stanici DP2 za poslední hodinu

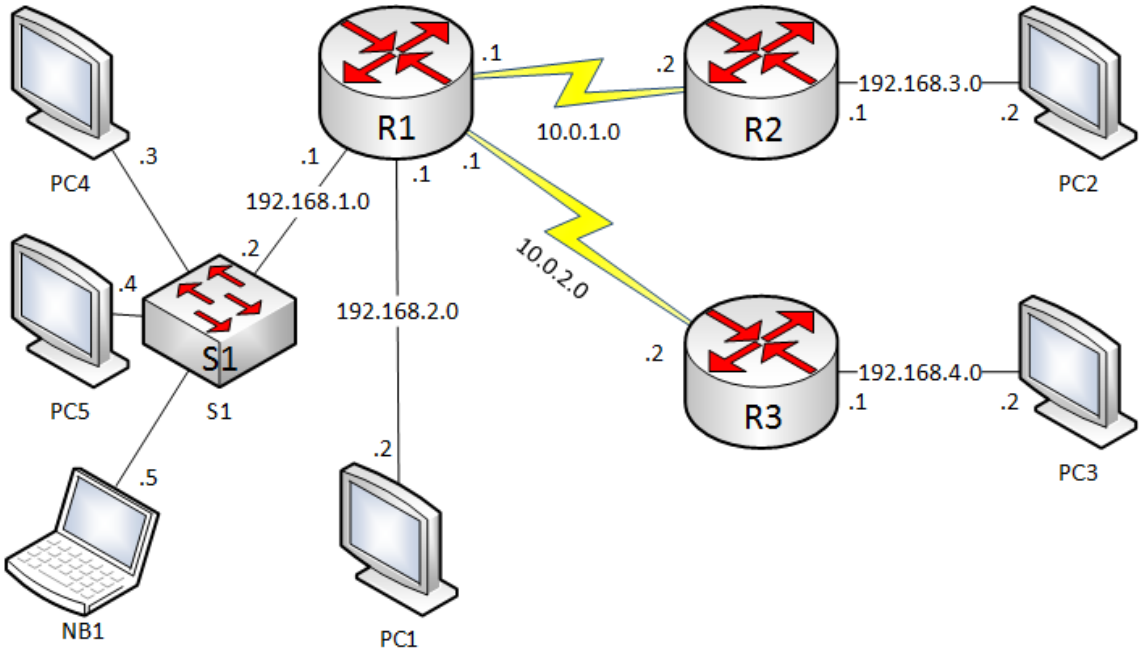
4.3 Sběr informací z jiných zařízení v počítačové síti

V posledních měřeních jsem se přesunul do školní laboratoře EB215, kde jsem měl k dispozici mnoho síťových prvků. Pro mé použití mi vyhovovaly zařízení značky Cisco. Cisco patří mezi přední výrobce v oblasti síťových zařízení. Jako koncové stanice mi posloužily počítače s OS Ubuntu, kterými je laboratoř taktéž vybavena.

Nejprve jsem si vytvořil vhodnou topologii sítě (obrázek 4.8), na které jsem chtěl monitorovat pomocí protokolu SNMP jeden router a jeden switch. Následně jsem všem rozhraním síťových prvků přidělil IP adresy. Dalším krokem bylo fyzické zapojení všech prvků a jejich konfigurace.

Na stanicích (PC1 - PC5, NB1) stačilo vždy jen nakonfigurovat dle schématu statickou IP adresu spolu s maskou a výchozí bránu. Nastavení routerů a switche si přiblížíme v samostatných podkapitolách. Na stanici PC1 bylo potřeba ještě nainstalovat pomocí pří-

kazu `# apt-get install` balíčky `snmp, rrdtool, librrdsperl, cron`, jelikož tato stanice sloužila jako hlavní stanice pro sběr dat.



Obrázek 4.8: Schéma zapojení sítě, na které bylo prováděno laboratorní měření

4.3.1 Nastavení routeru Cisco 2800

Na všech routerech (R1, R2 a R3) bylo potřeba pro všechna použitá rozhraní nastavit IP adresy spolu s maskami (u sériových linek na straně DCE i clock rate). Příklad nastavení pro sériové rozhraní na routeru R1 a směrovacího protokolu RIP (Routing Information Protocol) na routeru R2:

```
R1#configure terminal
R1(config)#interface  serial  0/1/0
R1(config-if)#ip address 192.168.1.1 255.255.255.0
R1(config-if)#clock rate  128000
R1(config-if)#no shutdown
R1(config)#exit

R2#configure terminal
R2(config)#router  rip
R2(config-router)#version 2
R2(config-router)#no auto-summary
R2(config-router)#network 192.168.3.0
R2(config-router)#network 10.0.1.0
R2(config)#exit
```

Výpis 9: Příkazy pro nastavení směrovacího protokolu RIP a rozhraní routeru

Na routeru R1, na kterém jsem chtěl sbírat data, bylo potřeba ještě nakonfigurovat SNMP server. Nejdříve je nutné se dostat do konfiguračního módu routeru a v něm zadat příkaz:

```
R1(config)#snmp-server community public RO
```

Tím se spustil server poskytující SNMP informace, jako „community string“ jsem nastavil řetězec `public` a `RO` je zkratka pro read-only, jelikož chci informace pouze číst. To je vše k nastavování routeru.

4.3.2 Nastavení switche Cisco Catalyst 3560

U tohoto switche bylo potřeba vytvořit virtuální rozhraní pro VLAN (Virtual Local Area Network) a nastavit mu IP adresu. Dále bylo potřeba přiřadit porty, které jsme chtěli monitorovat, do stejné, výše vytvořené VLAN. Třetí částí nastavování bylo zprovoznění SNMP serveru. Celou potřebnou konfiguraci najdete ve výpisu 10. Ve výpisu je pro stručnost uvedeno nastavení pouze pro jedno rozhraní (`FastEthernet 0/1`), ve skutečnosti jich ale bylo monitorováno dvanáct.

```
S1>enable
S1#configure terminal

S1(config)#interface  vlan  100
S1(config-if)#ip address 192.168.1.2 255.255.255.0
S1(config-if)#no shutdown
S1(config-if)#exit

S1(config)#interface  FastEthernet 0/1
S1(config-if)#switchport access vlan 100
S1(config-if)#switchport mode access
S1(config-if)#exit

S1(config)#snmp-server community public RO
```

Výpis 10: Příkazy pro nastavení rozhraní a SNMP serveru

4.3.3 Traffic ve směru IN na všech rozhraních routeru Cisco 2800

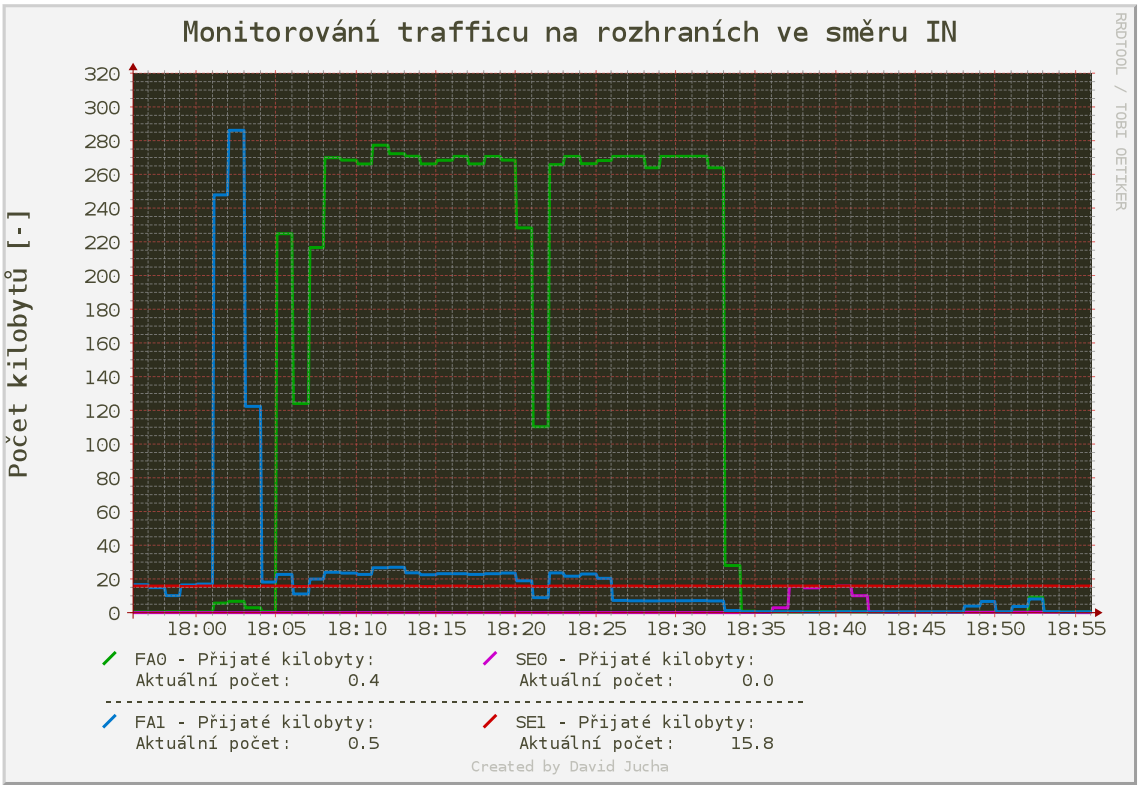
Prvním ze síťových prvků, na kterém jsem si vyzkoušel funkcionalitu protokolu SNMP a sběr informací pomocí tohoto protokolu, byl router firmy Cisco z řady 2800. Tento router byl osazen dvojicí FastEthernetových portů a také dvojicí sériových portů. Proto jsem se rozhodl využít všechny možné porty a monitorovat na nich provoz v obou možných směrech - jak ve směru IN, tak ve směru OUT (příloha C.3.1). Na schématu 4.8 se jednalo o router R1 a jako hlavní stanice pro sběr informací posloužil počítač označený jako PC1. Abych mohl data sbírat, bylo potřeba nejprve nakonfigurovat síťové prvky a tím zajistit jejich komunikaci (viz kapitola 4.3.1).

Po uvážení jsem si vytvořil skript, který mi monitoroval všechna rozhraní ve směru IN a následně mi z monitorovaných informací vytvořil grafy. Každý z vytvořených grafů ob-

sahoval 4 řady, odpovídající jednotlivým rozhraním. Graf zobrazující provoz za poslední hodinu ve směru download je na obrázku 4.9.

Pro získání informací o provozu byly použity následující příkazy:

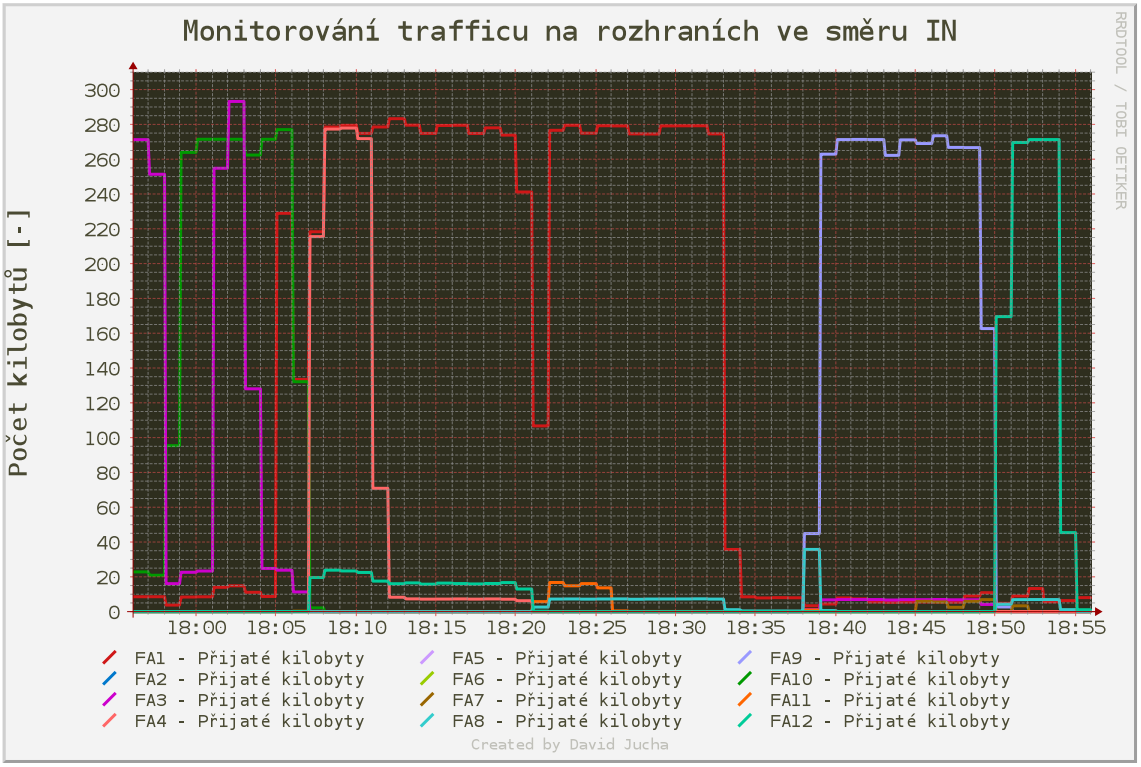
```
snmpwalk -v 2c -c public -Oqv 192.168.1.1
.1.3.6.1.2.1.2.2.1.10.1

snmpwalk -v 2c -c public -Oqv 192.168.1.1
.1.3.6.1.2.1.2.2.1.10.2

snmpwalk -v 2c -c public -Oqv 192.168.1.1
.1.3.6.1.2.1.2.2.1.10.3

snmpwalk -v 2c -c public -Oqv 192.168.1.1
.1.3.6.1.2.1.2.2.1.10.4
```



Obrázek 4.9: Traffic ve směru IN na všech rozhraních routeru Cisco 2800 za poslední hodinu

4.3.4 Traffic ve směru IN na rozhraních switche Cisco Catalyst 3560

Druhým síťovým prvkem, zvoleným pro monitorování, byl switch Cisco Catalyst 3560. Jednalo se o switch s 24 FastEthernetovými porty a dvěma GigabitEthernetovými porty. Zde jsem se rozhodl monitorovat provoz v obou směrech na prvních dvanácti FastEthernetových portech. Síť byla zapojena opět podle schématu 4.8 a switch (na obrázku označen jako S1) byl nakonfigurován tak, jak bylo uvedeno v kapitole 4.3.2. I zde posloužil počítač PC1 jako hlavní bod pro sběr informací.



Obrázek 4.10: Traffic ve směru IN na dvanácti rozhraních switche Cisco Catalyst 3560 za poslední hodinu

V grafu (obrázek 4.10) je zobrazeno najednou 12 řad, které odpovídají jednotlivým portům. Pro každý port byla zvolena jiná barva, aby bylo na první pohled vidět, o který port se jedná.

Přesné příkazy pro sběr informací o provozu na prvních dvanácti portech ve směru IN mají následující tvar (mění se akorát poslední pětičíslí v rozmezí 10001 až 10012):

```
snmpwalk -v 2c -c public -Oqv 192.168.1.2
.1.3.6.1.2.1.2.2.1.10.10001
```

Závěr

Cílem celé diplomové práce bylo naprogramování vhodných skriptů pro sběr systémových a síťových informací s použitím nástroje RRDtool. Sběr různých informací je tak rozsáhlý, že v podstatě na každé zařízení, ze kterého se dají získávat data, by se dal vytvořit text v rozsahu několika desítek stran. Cílem práce tedy nebylo se zaměřit na jedno konkrétní zařízení a zkoumat, které informace nám dokáže poskytnout, ale vytvořit různé situace, které mohou nastat v reálných podmínkách. A právě v těchto podmínkách se zaměřit na, z mého hlediska nejpoužívanější, sadu informací, které je možné z těchto zařízení získávat a je potřebné je z hlediska správců serverů či internetových providerů monitorovat a vyvozovat z nich důležité závěry.

Cílem práce bylo seznámit čtenáře se základy protokolu SNMP, který byl v mnoha měřeních využíván jako protokol pro sběr informací, poskytovaných ostatními stanicemi (PC, servery, notebooky) a zařízeními (routery a switche značky Cisco).

V další kapitole seznámila čtenáře se skriptovacími programovacími jazyky, kde především jazyk Perl byl využíván ve všech vytvořených skriptech, a s nástroji Cron a Lm-sensors, který slouží pro získávání doplňkových informací, jako jsou například teploty, napětí a rychlosti otáček ventilátorů.

V následující kapitole jej seznámila s nástrojem RRDtool a jeho hlavními funkcemi Create, Update a Graph, bez kterých jsme se ve všech měřeních neobešli. Zde je umístěn základní popis všech funkcí spolu s krátkými příklady. Přehled parametrů jednotlivých funkcí je v příloze práce.

V praktické části byly vytvořeny různé scénáře, ve kterých bylo provedeno vždy několik ukázkových měření. V první části se jednalo o lokální sběr informací ze stanice pomocí nástroje Lm-sensors, v druhé se jednalo o sběr informací pomocí vestavěných nástrojů OS Ubuntu. V dalších dvou částech byl využíván protokol SNMP, kde se nejprve s jeho pomocí získávaly informace se stanic, zapojených v sítí a následně také ze zařízení, zapojených v síti. První dvě části měření probíhaly v domácích podmínkách na mém osobní stolním PC, ve třetí části jsem využil možností třech školních virtuálních serverů a čtvrtá část měření probíhala ve školní laboratoři EB215 na zde umístěných síťových zařízeních.

Přínosem pro mou osobu bylo, že jsem si mohl prakticky vyzkoušet, jak probíhá monitorování různých informací, poskytovaných různými zařízeními s pomocí nástroje, který využívá k monitorování několik institucí. Při zpracovávání této práce jsem se dověděl mnoho nových a zajímavých informací, které změnily můj pohled na to, co a z jakého důvodu monitorovat.

Například souhrnné monitorování teplot, kde jsem monitoroval teploty na základní desce, procesoru a grafické kartě, bylo pro mě zajímavé. Překvapilo mě, že i při různém vytěžování sestavy během dvanácti hodin nepřesahovaly teploty přes sebe. Obzvlášť mě překvapilo, že teplota procesoru po celou dobu monitorování nepřekročila ostatní teploty, což bych před monitorováním samotným určitě netvrdil.

Dalším zajímavým monitorováním bylo monitorování otáček ventilátorů, umístěných na procesoru, ve zdroji napájení a ve skříni počítače. Zde mě překvapilo to, že hodnoty rychlosti ventilátorů ve skříni počítače a ve zdroji se po celou dobu měření prakticky

neměnily oproti ventilátoru na procesoru, který své rychlosti přizpůsoboval aktuální zátěži počítače.

Zajímavé by mohlo být monitorovat například teploty jednotlivých síťových prvků. Na těch, na kterých jsem prováděl svá měření já, nebylo možné informace o teplotách získat a to z důvodů nekompatibilní verze systému IOS těchto zařízení. Pro malé firmy, využívající vlastní NAS servery s podporou SNMP by mohlo být zajímavé monitorovat aktuální využití jejich diskového prostoru. S nástupem internetu věcí budou možnosti monitorování různých věcí jen stoupat.

Za celou práci jsem naprogramoval přes dvacet skriptů ve skriptovacím programovacím jazyce Perl, v každém skriptu bylo vytvářeno šest výstupních grafů, což nám dohromady dává přes 120 grafů. Průměrná délka jednotlivých skriptů se pohybovala okolo 190 řádků. Jednotlivá měření probíhala v délce od jedné hodiny až po jeden měsíc.

Součástí diplomové práce je i několik příloh, které by z důvodu jejich rozsahu narušovaly strukturu hlavní části práce. V těchto přílohách jsou popsány parametry hlavních funkcí nástroje RRDtool, je zde vložen i ukázkový skript pro sběr informací. Následně jsou zde umístěny další měření z každé, mnou definované oblasti, sada dlouhodobějších grafů ze všech měření a sada návodů pro praktickou výuku.

Existuje mnoho automatických nástrojů s grafickým uživatelským rozhraním, které vnitřně využívají nástroj RRDtool. Jedná se například o programy Cacti, Nagios, Munin a další. Právě srovnání a vyzkoušení těchto programů by mohlo být zajímavým doplněním této práce. Na vyzkoušení a srovnání výhod a nevýhod těchto nástrojů již nezbylo z hlediska rozsahu místo. Dalším možným rozšířením této práce by mohlo být zaměření se na jedno konkrétní zařízení, ze kterého by se získalo co nejvíce zajímavých informací a vytvořila se pro něj sada skriptů. Taktéž by pro prezentaci všech grafů mohla být vytvořena webová prezentace zobrazující všechny souhrnné grafy pro dané zařízení.

Bc. David Jucha

5 Literatura

[1] MAURO, Douglas R a Kevin J SCHMIDT. *Essential SNMP.* 2nd ed. Beijing: O'Reilly, 2005, 442 s. ISBN 05-960-0840-6.

[2] ALBING, Carl, J VOSSEN a Cameron NEWHAM. *Bash cookbook.* 1st ed. Sebastopol, CA: O'Reilly, c2007, xxi, 598 p. ISBN 978-059-6526-788.

[3] NEWHAM, Cameron a Bill ROSENBLATT. *Learning the bash Shell.* 3rd ed. Sebastopol, [Calif.]: O'Reilly, c2005, xvi, 333 p. ISBN 05-960-0965-8.

[4] CHRISTIANSEN, Tom, Brian D FOY, Larry WALL. *Programming Perl.* 4th ed. Sebastopol: O'Reilly, c2012, xli, 1130 p. ISBN 978-059-6004-927.

[5] CHRISTIANSEN, Tom a Nathan TORKINGTON. *Perl cookbook.* 2nd ed. Sebastopol, Calif.: O'Reilly, c2003, xxxiv, 927 p. ISBN 05-960-0313-7.

[6] SCHWARTZ, Randal L, Brian D FOY a Tom PHOENIX. *Learning Perl.* 6th ed. Sebastopol: O'Reilly, c2011, xxi, 363 p. ISBN 978-144-9303-587.

[7] BLANK-EDELMAN, David N. *Automating system administration with Perl.* 2nd ed., Rev. Cambridge: O'Reilly, c2009, xxiii, 639 p. ISBN 05-960-0639-6.

[8] BEAZLEY, David M a Brian K JONES. *Python cookbook.* 3nd ed. Beijing: O'Reilly, 2013, xvi, 687 s. ISBN 978-1-449-34037-7.

[9] LUBANOVIC, Bill. *Introducing python: modern computing in simple packages.* S.l.: O'Reilly Media, Inc, Usa, 2013. ISBN 978-144-9359-362.

[10] *FilePermissions: Community Help Wiki.* [online]. [cit. 2015-02-01]. Dostupné z: https://help.ubuntu.com/community/FilePermissions

[11] *CronHowto: Community Help Wiki.* [online]. [cit. 2015-02-01]. Dostupné z: https://help.ubuntu.com/community/CronHowto

[12] Frodo Looijaard. *Lm-sensors.* [online]. [cit. 2015-02-01]. Dostupné z: http://www.lm-sensors.org/

[13] OETIKER, Tobias. *About RRDtool.* [online]. [cit. 2015-02-01]. Dostupné z: http://oss.oetiker.ch/rrdtool/

[14] OETIKER, Tobias. *Tobi Oetiker: Tobi Oetiker's Toolbox.* [online]. [cit. 2015-02-01]. Dostupné z: https://tobi.oetiker.ch/hp/

[15] OETIKER+PARTNER AG. *OETIKER+PARTNER AG: Team.* [online]. [cit. 2015-02-01]. Dostupné z: http://www.oetiker.ch/

[16] OETIKER, Tobias. *RRDtool - RRDtool Documentation.* [online]. [cit. 2015-02-01]. Dostupné z: https://oss.oetiker.ch/rrdtool/doc/index.en.html

Adresářová struktura přiloženého CD

Cesta	Obsah
/Grafy 4.1	- Složka obsahující grafy odpovídající kapitole 4.1 a C.1
/Grafy 4.2	- Složka obsahující grafy odpovídající kapitole 4.2 a C.2
/Grafy 4.3	- Složka obsahující grafy odpovídající kapitole 4.3 a C.3
/RRD databáze 4.1	- Složka obsahující databáze RRD odpovídající kapitole 4.1 a C.1
/RRD databáze 4.2	- Složka obsahující databáze RRD odpovídající kapitole 4.2 a C.2
/RRD databáze 4.3	- Složka obsahující databáze RRD odpovídající kapitole 4.3 a C.3
/Skripty 4.1	- Složka obsahující skripty odpovídající kapitole 4.1 a C.1
/Skripty 4.2	- Složka obsahující skripty odpovídající kapitole 4.2 a C.2
/Skripty 4.3	- Složka obsahující skripty odpovídající kapitole 4.3 a C.3
/JUC0010.pdf	- Soubor obsahující text diplomové práce

A Popis parametrů funkcí nástroje RRDtool

A.1 Popis parametrů funkce CREATE

V této kapitole se budeme věnovat jednotlivým parametrům funkce CREATE. Příkaz začíná klíčovými slovy **`rrdtool create`** a pokračuje následujícími parametry:

- `filename` - nám udává název vytvářené databáze, který by měl končit koncovkou `.rrd`. Není to však nutnou podmínkou, RRDtool si poradí s jakýmkoliv názvem. Tento název může obsahovat i cestu, kde se má vytvářená databáze uložit. Název tedy může být například `/home/ubuntu/Documents/rrd/test.rrd`, `test.rrd` nebo pouze `test`.
- **`--start|-b`** `start time` - nám udává čas pro přidání prvního záznamu do databáze. Zadává se v sekundách uplynulých od 1.1.1970 UTC (Coordinated Universal Time). Výchozím nastavením pro tento čas je `(now - 10s)`, což neznamená nic jiného, než aktuální čas před deseti sekundami.
- **`--step|-s`** `step` - nám udává délku kroku v sekundách. Po každém takovém kroku se zapíše do databáze jeden záznam. Výchozí hodnotou je zde 300 sekund.
- **`--no-overwrite`** - zajišťuje, abychom si nepřepsali již dříve vytvořenou databázi se stejným názvem.

A.1.1 Parametry DS

- **`DS:`**`ds-name:DST:dst arguments` - DS (Data Source) lze chápat jako proměnnou, která se ukládá do databáze. Těchto proměnných může být v databázi obsaženo hned několik. Pro každý DS ukládaný do databáze musíme definovat jeho parametry:
  - `ds-name` - jedná se o jméno, které specifikuje daný DS. Název musí mít minimálně jeden a maximálně 19 znaků a může obsahovat znaky [a-zA-Z0-9_].
  - `DST` - DST (Data Source Type) nám definuje typ datového zdroje. Ostatní parametry DS závisí na zvoleném DST. Pro GAUGE, COUNTER, DERIVE a ABSOLUTE má tvar:
    `GAUGE|COUNTER|DERIVE|ABSOLUTE:heartbeat:min:max`

    Pro COMPUTE:
    `COMPUTE:rpn-expression`
    - `GAUGE` - je používán nejčastěji, do databáze zapisuje přímo hodnoty, které jsme získali. Příkladem může být počet přihlášených uživatelů nebo počet spuštěných procesů.
    - `COUNTER` - umí zaznamenávat kontinuální zvyšování hodnoty za jednotku času. Typickým příkladem může být počet přijatých/odeslaných paketů na síťovém rozhraní. Hodnoty COUNTERu se nikdy nesnižují, výjimkou je pouze stav, kdy COUNTER přeteče.

    * DERIVE - ukládá hodnotu, vzniklou vždy z posledního a aktuálního bodu DS. Vnitřně pracuje jako COUNTER bez kontroly přetečení a na rozdíl od něj může nabývat záporných hodnot.
    * ABSOLUTE - se používá pro čítače, které jsou resetovány po načtení hodnoty.
    * COMPUTE - umožňuje ukládat výsledky vzorce (definovaného v jeho parametru), který aplikujeme na jiné DS databáze RRD. Slouží tedy pro výpočet hodnot z jiného DS. Oproti DS nepotřebuje získávat data pro ukládání, jelikož si je sám počítá.
  - heartbeat - nám definuje maximální počet sekund, které mohou uběhnout mezi dvěma aktualizacemi DS. Jinak se po uplynutí této doby přiřadí DS neznámá hodnota (UNKNOWN).
  - min, max - nám definuje očekávaný vstupní rozsah hodnot od minima po maximum pro data dodávané DS. Pokud poskytnutá data nespadají do definovaného intervalu ⟨min;max⟩ nabudou opět neznámé hodnoty (UNKNOWN). Tím, že zadáme minimální/maximální vstupní hodnoty, umožníme nástroji RRDtool alespoň základní kontrolu vstupních dat a zamezíme případnému vložení nereálných hodnot. Jestliže neznáme velikost očekávaných hodnot, můžeme jako hodnotu min/max zadat neznámou hodnotu U (UNKNOWN).
  - rpn-expression - nám definuje vzorec, potřebný pro výpočet hodnot tohoto DS. Ten je získá výpočtem z jiného DS stejné RRD.

A.1.2 Parametry RRA

- **RRA:**CF:cf arguments - v databázi RRD jsou jednotlivé informace uloženy v RRA (Round Robin Archives). Každý archiv RRA je složen z množství dat a statistických hodnot pro každý definovaný DS a je definován jedním RRA řádkem. Když vložíme záznam do databáze RRD, je neprodleně přiřazen časové pozici o maximální délce definované parametrem -s. Sloučením tohoto záznamu a časové pozice vznikne primární datový bod.

  Jednotlivými RRA definujeme počet uchovávaných hodnot na daný časový úsek. Chceme-li vytvářet týdenní grafy, měli bychom uchovávat více hodnot, než u dlouhodobějších grafů, abychom jej měli detailnější. Pro ta samá data chceme vytvořit i měsíční graf, tady už ale nepotřebujeme tolik týdenních hodnot, jelikož nám stejně ve výsledném grafu zaniknou. RRA nám umožňují ukládat různá data pro různé časové úseky.

  - CF - data jsou zpracovávána konsolidační funkcí (CF - Consolided Function) archivu. Existuje několik konsolidačních funkcí, které konsolidují primární datové body s agregačními funkcemi: AVERAGE, MIN, MAX, LAST.
    * AVERAGE - průměrná hodnota dat je ukládána
    * MIN - minimální hodnota dat je ukládána

    * `MAX` - maximální hodnota dat je ukládána
    * `LAST` - poslední hodnota dat je ukládána

  Avšak je důležité si uvědomit, že agregací dat můžeme zvyšovat jejich nepřesnost. Proto musíme zvážit, jak máme danou agregační funkci použít, abychom potřebná data uchovali i po procesu agregace. Formát RRA řádku má následující tvar:
  `RRA:AVERAGE|MIN|MAX|LAST:xff:steps:rows`

- `xff` - XFF (Xfiles Factor) - jedná se o číslo v intervalu $\langle 0;1 \rangle$, které nám určuje, jak velké procento konsolidačního intervalu může obsahovat neznámé (`UNKNOWN`) hodnoty, aby byla konsolidační hodnota uznána jako známá. V podstatě jde o poměr neznámých hodnot k celkovému počtu hodnot v daném intervalu.
- `steps` - celkový počet hodnot, tvořících daný vzorek, na který se následně aplikuje konsolidační funkce, ze které se výsledná hodnota uloží do databáze.
- `rows` - výsledný počet řádků (hodnot), které chceme v databázi uchovávat. Tato hodnota musí být větší než 1.

Kapitola A.1 byla převzata a přeložena z oficiální anglické dokumentace nástroje RRDtool, která je umístěná na jeho domovských stránkách [16].

A.2 Popis parametrů funkce UPDATE

- `filename` - zde zadáváme název databáze RRD, do které chceme vkládat data. Opět se může jednat o název spojený s cestou k dané databázi nebo jen o název samotný.

- **`--template|-t`** `ds-name[:ds-name]...` - ve výchozím nastavení funkce UPDATE očekává, že data, která chceme vkládat do databáze RRD, jsou v takovém pořadí, jak jsou definována v jednotlivých DS, vyjma DS COMPUTE. U COMPUTE se totiž počítá s tím, že například druhý DS je typu COMPUTE a vypočítává si data ze třetího DS. Takové zadávání je příliš náchylné na chyby, a proto vznikl parametr `template`. S jeho pomocí můžeme definovat, v jakém pořadí a do jakých DS chceme zapisovat data. V případě, že v `template` zadáme neexistující DS, zápis neproběhne a skončí chybovým hlášením. I když to na první pohled vypadá, že bychom pomocí `template` mohli do databáze vkládat data asynchronně, není tomu tak. RRDtool si implicitně u DS (vyjma DS COMPUTE) k hodnotám neobsažených v `template` přiřazuje neznámou (`UNKNOWN`) hodnotu. Do DS COMPUTE bychom neměli pomocí `template` vkládat data, jelikož si je samy vypočítávají. Jestliže se tak nedopatřením někdy stane, RRDtool je stejně bude ignorovat.

- **`--daemon`** `address` - v případě zadání tohoto parametru se RRDtool zkusí připojit k démonovi `rrdcached`. V případě úspěšného připojení budou hodnoty zaslány přímo tomuto démonovi namísto přímého vkládání do databáze RRD.

- **N**|timestamp:value[:value...] - Každá hodnota (value), kterou chceme umístit do databáze, by měla nést časovou značku (timestamp). Tato časová značka může být definována buď jako čas v sekundách uplynulých od 1.1.1970 nebo použitím písmene „N“, které odkazuje na aktuální čas. Záporné hodnoty času jsou odečítány od času aktuálního. Při jejich použití ale musí být oddělen parametr a jeho hodnota dvěma pomlčkami (--), jinak by byl čas parsován jako parametr. K časové značce přikládáme i všechny hodnoty (values), které chceme umísťovat do jednotlivých DS. Pořadí zadávaných hodnot musí odpovídat DS, definovaných v jednotlivých archivech RRA, nebo musí dodržovat šablonu, definovanou v parametru template. Když neznáme nějakou hodnotu, můžeme místo ní použít písmeno U (např.: N:0.5:1:U:2). Ve většině případů jsou zpracovávány číselné hodnoty, existují však i moduly, které umožňují vlastní parsování. I zde slouží jako oddělovače jednotlivých výrazů dvojtečky.

Kapitola A.2 byla převzata a přeložena z oficiální anglické dokumentace nástroje RRDtool, která je umístěná na jeho domovských stránkách [16].

A.3 Popis parametrů funkce GRAPH

- filename - jméno nebo cesta se jménem grafu, který generujeme. Doporučené formáty souboru pro grafy jsou *.png, *.svg nebo *.eps.

A.4 Parametry OPTIONS

A.4.1 Časový rozsah

[**-s|--start** time] [**-e|--end** time] [**-S|--step** seconds]

V parametrech časového rozsahu volíme velikost oblasti (mezi parametrem start a end), kterou chceme ve výsledném grafu vykreslit. Ve většině případů chceme zobrazovat graf za poslední dobu (hodinu, den, týden, měsíc) do aktuálního času. Defaultně je nastaveno zobrazování grafu za uplynulý den v nejlepším možném rozlišení. Rozlišení grafu závisí na použitém RRA, který si RRDtool vybírá sám. To však můžeme změnit pomocí parametru step. Chceme-li si například zobrazit graf s hodinovým rozlišením, nastavíme parametr step na hodnotu 3600. Časové hodnoty můžeme zadávat ve více formátech. Buď v době uplynulé od 1.1.1970 nebo můžeme používat časové jednotky (now, second(s), minute(s), hour(s), day(s), week(s), month(s), year(s)).

A.4.2 Nadpisy

[**-t|--title** string] [**-v|--vertical-label** string]

Parametr title umožňuje vytvořit horizontální nadpis pro graf, umístěný nahoře na středu. Obdobně funguje i parametr vertical-label, který umisťuje nadpis vertikálně na levou stranu grafu.

A.4.3 Velikost

[**-w|--width** pixels] [**-h|--height** pixels] [**-j|--only-graph**]
[**-D|--full-size-mode**]

Pomocí parametru `width` a `height` můžeme specifikovat velikost vytvářeného grafu v pixelech. Tato velikost definuje velikost oblasti grafu, tudíž výsledné rozměry jsou větší o oblasti nadpisů a legend. Defaultní hodnoty jsou nastaveny na 400x100 pixelů.

Parametr `only-graph` slouží pro vykreslení miniatury grafu, která může sloužit jako náhled či ikona. Všechny popisky jsou z grafu odstraněny a předpokládají se zde malé rozměry. Když zadáme parametr `full-size-mode`, budou výsledné rozměry grafu i s nadpisy a legendou odpovídat velikosti, definované v parametrech `width` a `height`.

A.4.4 Limity hodnot

[**-u|--upper-limit** value] [**-l|--lower-limit** value]
[**-r|--rigid**]

V defaultním nastavení se automaticky nastavuje rozsah hodnot osy Y v závislosti na datech, která má zobrazovat. Toto chování lze změnit pomocí parametru `rigid`, kterým přinutíme zobrazovat data v námi zadaném rozsahu, definovaném parametry `upper-limit` a `lower-limit`.

[-A|--alt-autoscale]

Výchozí algoritmus pro určování rozsahu hodnot osy Y někdy nepracuje spolehlivě. Sasha Mikheev vyvinul lepší algoritmus, který vypočítává minimální a maximální hodnoty osy Y z aktuálních minimálních a maximálních hodnot. Tento algoritmus můžeme povolit pomocí parametru `alt-autoscale`.

[-J|--alt-autoscale-min] [-M|--alt-autoscale-max]

Parametr `alt-autoscale` určuje najednou minimální i maximální hodnoty. Při použití parametru `alt-autoscale-min` se určuje jen minimální hodnota, maximální hodnota, pokud není jinak definováno, bude nulová. U parametru `alt-autoscale-max` je tomu naopak.

[-N|--no-gridfit]

Abychom zabránili rozmazávání grafů, které způsobuje anti-aliasing, RRDtool umí přichytit body k pixelům našeho zařízení. Výsledkem je ostřejší vzhled grafů. Jestliže se nám tento ostřejší vzhled nelíbí, můžeme jej pomocí parametru `no-gridfit` vypnout. Grid-fitting je defaultně vypnutý pro tyto typy souborů: `*.pdf`, `*.eps`, `*.svg`.

A.4.5 Osa X

[**-x|--x-grid** GTM:GST:MTM:MST:LTM:LST:LPR:LFM]
[**-x|--x-grid** none]

Popisky osy X jsou dosti složité na konfiguraci. Jestliže nemáme speciální požadavky na jejich úpravy, je nejlepší možností nechat je nakonfigurovat automaticky. Pro smazání popisků osy X spolu s vertikálními linkami zadáme v parametru `x-grid` řetězec `none`.

Mřížka grafu je definována zadáním určitého času na ?TM pozicích. Můžeme si vybrat z `SECOND`, `MINUTE`, `HOUR`, `DAY`, `WEEK`, `MONTH`, `YEAR`. Následně definujeme číselnou hodnotu, která udává dobu mezi jednotlivými linkami nebo popisky. Například si zvolíme MINUTE:10, což znamená, že se nám vykreslí vertikální linky co 10 minut. Tuto dvojici (?TM:?ST) musíme definovat hned třikrát - pro základní mřížku (G??), pro hlavní mřížku (M??) a pro popisky (L??). Pro popisky však ještě musíme definovat umístění (LPR) a formátovací řetězec *strftime* (LFM). LPR definuje, kde má být každý popisek umístěn. Jestliže zadáme nulu, bude umístěn přímo pod odpovídající linku. Jinak můžeme zadat dobu v sekundách a tím způsobíme, že se daný popisek posune o tento námi zadaný interval. Více informací o jednotlivých formátech řetězce *strftime* nalezneme přímo v manuálových stránkách po zadání příkazu `# man strftime`.

A.4.5.1 Příklad nastavení osy X Následujícím složeným parametrem nastavíme, aby se nám hlavní mřížka vykreslovala každou hodinu (`HOUR:1`), základní každých pět minut (`MINUTE:5`) a popisky každé dvě hodiny (`HOUR:2`). Popisky budou umístěny přímo pod hlavní mřížku (`0`) a budou se formátovat podle řetězce (`%R`), který nám naformátuje čas ve formátu HH:MM.

```
--x-grid MINUTE:5:HOUR:1:HOUR:2:0:%R
```

A.4.6 Osa Y

```
[-y|--y-grid grid step:label factor]
  [-y|--y-grid none]
```

Horizontální linky se na ose Y zobrazují spolu s každým krokem na zaznamenávaném intervalu. Popisky jsou umístěny na každé hlavní lince. Obdobně jako u osy X můžeme v parametru `y-grid` zadat hodnotu `none`, abychom zakázali vykreslení horizontálních linek spolu s popisky. Neodstraní se jen popisky, ale i jejich rezervované místo. Pokud chceme z nějakého důvodu toto místo ponechat, můžeme to provést manuálně pomocí parametru `units-length`. Defaultní nastavení parametru `y-grid` odpovídá automatickému zvolení rozumných hodnot.

```
[--left-axis-format format-string]
```

Defaultně je formát popisků osy vybrán automaticky. V případě, že jej chceme sami definovat, můžeme použít tento parametr spolu se stejnými `%lf` argumenty, které známe z příkazů `PRINT` či `GPRINT`.

```
[-Y|--alt-y-grid]
```

Umístění mřížky osy Y je dynamické - je založeno na rozsahu osy Y. Vestavěný algoritmus zajišťuje, aby bylo linek v mřížce dostatek a aby jich naopak nebylo málo. Také

dohlíží na to, aby linky dodržovaly metriku - to znamená, že jsou linky umísťovány každých 1,2,5 nebo 10 jednotek. Tento parametr zajišťuje, aby nebylo zobrazováno příliš mnoho desetinných míst v popisku, když nám má graf například zobrazit hodnoty od 69,998 do 70,001.

`[-o|--logarithmic]`

Tímto parametrem povolíme na ose Y používání logaritmického měřítka.

`[-X|--units-exponent value]`

Běžně se hodnoty popisků zobrazují v příslušných jednotkách (m, k, M atd.). Někdy ale můžeme chtít, aby se nám nezobrazovaly tyto zkratky, ale aby se nám zobrazovaly celé hodnoty. Chceme-li například na ose Y zobrazovat hodnoty v jednotkách k (kilo) = 1000 = 10e3, zadáme do parametru `units-exponent` hodnotu exponentu (`value`) `3`. Zadávaná hodnota je typu `integer`, musí být násobkem tří a může nabývat hodnot z intervalu $\langle -18; 18 \rangle$. Při zadání nulové hodnoty zamezíme změně měřítka osy Y.

`[-L|--units-length value]`

Pomocí parametru `units-length` můžeme zadat délku, kterou budou obsahovat popisky osy Y. V některých případech nemusí být dostačující automaticky zvolená délka popisku.

`[--units=si]`

Pomocí tohoto parametru povolíme konverzi logaritmických hodnot grafu na odpovídající jednotky (m, k, M atd.). Defaultně se používá SI notace.

A.4.7 Pravá osa Y

`[--right-axis scale:shift]`

Druhá osa Y může být vykreslena na pravé straně grafu. Je vázána k levé ose pomocí parametru `scale` a `shift`. `Scale` udává posun hodnot osy Y o zadaný násobek, `shift` udává posun hodnot osy Y o zadaný počet jednotek.

`[--right-axis-label label]`

Pomocí parametru `right-axis-label` můžeme definovat popisek pravé osy Y.

`[--right-axis-format format-string]`

Stejně jako u základní osy Y tak i u pravé osy Y je defaultně formát popisků osy vybírán automaticky. Pro případnou změnu tohoto formátu můžeme použit tento parametr spolu se stejnými `%lf` argumenty, které známe z příkazů `PRINT` či `GPRINT`.

A.4.8 Legenda

`[-g|--no-legend]`

Zakáže tvorbu legendy pro graf. Vykreslí se pouze graf.

`[-F|--force-rules-legend]`

Vynutí generování HRULE a VRULE legend pouze v případě, že HRULE nebo VRULE nebude vykreslena v grafu z důvodu definovaných hranic.

`[--legend-position=(north|south|west|east)]`

Výběr pozice legendy na určenou stranu grafu. Ve výchozím nastavení se umísťuje na jih (`south`). Při umístění na západní (`west`) nebo východní (`east`) stranu je nutné přidat znaky pro zalamování řádků manuálně.

`[--legend-direction=(topdown|bottomup)]`

Umístění položek legendy v zadaném vertikálním pořadí. Defaultně je zvoleno shora-dolů (`topdown`).

A.4.9 Ostatní možnosti

`[-c|--color COLORTAG#rrggbb[aa]]`

Pomocí parametru `color` můžeme měnit barvy jednotlivých elementů vykreslovaného grafu. `COLORTAG` nám určuje element, pro který chceme měnit barvu. Všechny elementy jsou v tabulce A.1. Každá výsledná barva je složena ze tří dvojic hexadecimálních čísel, specifikujících jednotlivé složky RGB (00 je nejsvětlejší a FF nejtmavší). Ještě je zde ale jedna možnost. Můžeme si zvolit i průhlednost dané barvy (FF je neprůhledná). Tato volba ale není povinná. Například pro červené neprůhledné pozadí zvolíme parametr následovně: `color BACK#FF0000`.

Tabulka A.1: Seznam podporovaných COLORTAGů

COLORTAG	Význam COLORTAGu
BACK	Barva pozadí okolo grafu
CANVAS	Barva oblasti pro graf
SHADEA	Barva pro levý a horní okraj
SHADEB	Barva pro pravý a dolní okraj
MGRID	Barva pro hlavní mřížku
GRID	Barva pro vedlejší mřížku
FONT	Barva fontů
AXIS	Barva os grafu
FRAME	Barva ohraničení okolo barevných bodů v legendě
ARROW	Barva šipek na koncích os

`[--grid-dash on:off]`

Umožňuje upravovat vzhled vykreslovaných mřížek. Defaultně je mřížka vykreslována podle šablony `1on:1off`. Pomocí tohoto parametru si tuto vlastnost můžeme změnit na námi zvolenou. Například pro plné čáry bychom zvolili `grid-dash 1:0`.

[**--border** width]

Šířka 3D okraje výsledného obrázku v pixelech. Defaultní hodnota je 2. Nulová hodnota okraj zakáže. Pro nastavení barvy můžeme použít výše zmíněný COLORTAG SHADEA a SHADEB.

[**--dynamic-labels**]

Namísto barevných čtverečků vedle popisků v legendě zobrazí tvary, odpovídající vykreslovaným prvkům v grafu.

[**-m|--zoom** factor]

Umí několikanásobně přiblížit graf. Velikost přiblížení je dána faktorem. Ten musí nabývat hodnoty větší než 0.

[**-n|--font** FONTTAG:size:[font]]

Pro jednotlivé elementy grafu si můžeme zvolit různé fonty. Seznam všech podporovaných FONTTAGů nalezneme v tabulce A.2.

Tabulka A.2: Seznam podporovaných FONTTAGů

FONTTAG	Význam FONTTAGu
DEFAULT	Nastavuje defaultní hodnotu pro všechny elementy grafu
TITLE	Font pro hlavní titulek grafu
AXIS	Font pro popisky os
UNIT	Font pro vertikální titulky os
LEGEND	Font pro legendu grafu
WATERMARK	Font pro vodoznak na okraji grafu

Jestliže chceme použít font Times pro titulek, parametr font bude mít tento tvar:

```
--font TITLE:14:Times
```

V případě, že font obsahuje název s mezerami, je potřeba dát argument parametru do uvozovek:

```
--font „TITLE:14:Nejaky font“
```

Když neuvedeme název fontu, můžeme měnit pouze jeho velikost:

```
--font TITLE:14:
```

Když zvolíme velikost fontu 0, potom můžeme měnit typ fontů bez ohledu na jeho velikost. Tento parametr je velmi užitečný v případě, když chceme ovlivnit defaultní font bez změny nastavených velikostí:

```
--font DEFAULT:0:Times
```

[**-E|--slope-mode**]

Grafy jsou v RRDtool složeny z křivek, které mají ostré hrany, a skoky mezi jednotlivými hodnotami jsou kolmé přesně tak, jak je RRDtool počítá. Někteří lidé nemají rádi tyto ostré skoky a tak v RRDtool existuje parametr `slope-mode`, který je na pohled zjemní.

[**-a|--imgformat PNG|SVG|EPS|PDF**]

Formát souboru vygenerovaného obrázku obsahujícího graf. Pro vektorové formáty si můžeme vybrat z následujících standardních PostScriptových fontů: Courier-Bold, Courier-BoldOblique, Courier-Oblique, Courier, Helvetica-Bold, Helvetica-BoldOblique, Helvetica-Oblique, Helvetica, Symbol, Times-Bold, Times-BoldItalic, Times-Italic, Times-Roman a ZapfDingbats.

[**-T|--tabwidth** value]

Nastavuje velikost tabulátoru na námi definovanou velikost v pixelech. Defaultně je tato velikost nastavena na 40 pixelů.

[**-b|--base** value]

Jestliže vykreslujeme grafy paměti (a ne síťového provozu), tento parametr můžeme nastavit na hodnotu 1024, což znamená, že jeden kB je 1024 B. Pro měření provozu platí rovnost 1 kb/s = 1000 b/s.

[**-W|--watermark** string]

Přidá na dolní stranu grafu horizontálně centrovaný vodoznak, obsahující námi požadovaný řetězec (`string`).

A.5 Parametry DATA DEFINITION

DEF:<vname>=<rrdfile>:<ds-name>:<CF>[:step=<step>][:start=<time>]
[:end=<time>][:reduce=<**CF**>]

Pomocí této syntaxe načítáme data ze souboru RRD a předáváme je funkci graph, aby nám je zobrazila v grafu. Virtuální jméno `vname` může být použito po celou dobu trvání skriptu. Řetězec `rrdfile` nám udává název databáze RRD. Defaultně se vybírá ten RRA, který obsahuje správně zkonsolidovaná data v požadovaném rozlišení. Toto rozlišení může být přepsáno pomocí parametru `--step` nebo ještě opětovně přepsáno pomocí specifikování velikosti kroku `step`. Časové rozpětí těchto dat je defaultně nastaveno stejně, jako časové rozpětí vykreslovaného grafu. Pokud chceme toto rozpětí změnit, můžeme toho docílit specifikováním začátku (`start`) a konce (`end`). Zde ale musíme dát pozor na to, abychom při zadávání času správně použili tzv. escape sekvence.

Rozlišení grafu vychází z rozlišení dat. Ideálně by každý pixel grafu měl korespondovat s jedním datovým bodem z archivu RRA. Například máme databázi RRD s archivem RRA a s rozlišením dat 1800 sekund na jeden datový bod. Chceme vytvořit graf o šířce 400 pixelů, z toho nám vychází časové rozlišení 400*1800 sekund - to je 8 dní a 8 hodin. Při vytváření grafu si tedy zvolíme příslušný rozsah začátku a konce, kdy konec je například aktuální čas a začátek = konec - 8 dní a 8 hodin.

Když chceme používat konsolidační funkci, můžeme tak učinit i zde ve funkci `graph`. Ta se zde používá pro redukování hustoty dat a její chování můžeme změnit zadáním `:reduce=<CF>`. Tento parametr je volitelný a umožňuje definovat, která konsolidační funkce má být použita při redukování počtu dat.

A.5.1 Příklad definice dat

```
DEF:proc=pocetProcesu.rrd:proc:AVERAGE
DEF:procT=pocetProcesu.rrd:proc:AVERAGE:start=end-2h
DEF:procT=pocetProcesu.rrd:proc:AVERAGE:start=8\:00:end=start+2h
```

Výpis 11: Ukázky definice dat

A.6 Parametry DATA CALCULATION

CDEF : `vname=RPN expression`

Tento příkaz vytvoří novou sadu datových bodů (pouze v paměti, nikoliv v databázi RRD) z jedné nebo více sad těchto datových bodů. RPN instrukce jsou použity k vyhodnocování matematických funkcí pro každý datový bod. Výsledky `vname` mohou být dále použity ve skriptu tak, jako by byly generovány sekcí DEF. Pomocí sekce `DATA CALCULATION` obecně můžeme přepočítávat hodnoty. Typickým příkladem může být například přepočet mezi bity a bajty, kdy získávaná data jsou v bitech a v grafu chceme zobrazovat bajty nebo opačně.

A.6.1 Příklad počítání s daty

```
CDEF:datavbitech=data,8,*
```

Výpis 12: Ukázka počítání s daty

V tomto příkladu jsme měli v DS `data` uloženy hodnoty. V grafu jsme nechtěli zobrazovat hodnoty v bajtech, ale v bitech. Vytvořili jsme si nový DS nazvaný `datavbitech`, do kterého jsme uložili přepočítaná data z prvního DS. Znak `*` na konci znamená násobení a `8` je číselná hodnota, kterou jsme chtěli násobit hodnoty z původního DS.

A.7 Parametry VARIABLE DEFINITION

VDEF : `vname=RPN expression`

Tento příkaz nám vrací hodnotu a/nebo čas v závislosti na použitém RPN výrazu. Výsledkem `vname` bude, v závislosti na požité funkci, hodnota a časová značka. Když použijeme tento `vname` v jiném RPN výrazu, můžeme využívat jeho hodnoty, ale pouze v případě, že tyto hodnoty můžeme na dané místo vkládat. Proměnná `vname` může být použita v různých částech `graph` a `print` elementů. Nejčastěji se v RPN výrazech používá `MINIMUM`, `MAXIMUM`, `AVERAGE`, `FIRST`, `LAST` a `TOTAL`.

A.7.1 Příklad definice proměnných

```
VDEF:minimumProcesu=proc,MINIMUM
VDEF:maximumProcesu=proc,MAXIMUM
VDEF:prumerProcesu=proc,AVERAGE
```

Výpis 13: Ukázky definice proměnných

A.8 Parametr PRINT ELEMENT

PRINT:`vname:CF:format`

Nám umožňuje zpracovávat získávané hodnoty a jejich aktuální velikosti posílat na standardní výstup `STDOUT`. To je výhodné především pro ty, kteří chtějí volat funkce RRDtoolu z různých skriptů a chtějí v těchto skriptech zpracovávat tyto získané hodnoty. Získané hodnoty můžeme dále formátovat a to pomocí formátovacích řetězců.

A.9 Parametry GRAPH ELEMENT

GPRINT:`vname:CF:format`

Funkce `GPRINT` je totožná s výše uvedenou funkcí `PRINT` s tím rozdílem, že funkce `GPRINT` neposílá naměřené hodnoty na standardní výstup, ale vykresluje je přímo pod výsledný graf. Zde můžeme vytvářet různé sumarizace a souhrny měřených dat. I funkce `GPRINT` využívá formátovací řetězce.

COMMENT:`text`

Parametr `COMMENT` nám umožňuje vytisknout námi zadaný text do legendy grafu.

VRULE:`time#color[:legend][:dashes[=on_s[,off_s[,on_s,off_s]...]]`
`[:dash-offset=offset]]`

Pomocí parametru `VRULE` si můžeme do grafu umístit vertikální linku v zadaném čase (`time`). Barvu si můžeme opět volit pomocí tří dvojic hexadecimálních čísel reprezentujících barevné složky RGB, volitelně si můžeme upravit i alfa kanál, umožňující volit průhlednost zvolené barvy. Volitelně si můžeme zapnout i zobrazení popisku linky v legendě. Časem může být číslo nebo proměnná `VDEF`. Přerušované linky můžeme povolit v sekci `dashes`. Zde si můžeme definovat i styl těchto linek.

HRULE:`value#color[:legend][:dashes[=on_s[,off_s[,on_s,off_s]..]]`
`[:dash-offset=offset]]`

`HRULE` nám umožňuje vykreslit horizontální linku v grafu. Chová se jako `LINE` s tím rozdílem, že se nemění s měřítkem grafu. Ostatní parametry jsou stejné jako u výše uvedeného `HRULE`.

LINE[width]:`value[#color][:[legend][:STACK][:skipscale]]`
`[:dashes[=on_s[,off_s[,on_s,off_s]...]][:dash-offset=offset]]`

Jedná se o jeden z hlavních parametrů z této sekce, jelikož se používá pro vykreslení průběhu námi sledovaných hodnot. Vykreslí se tedy linka (`LINE`) o námi definované šířce. Šířka (`width`) může být číslo s desetinnými místy. Když si nezvolíme barvu, kterou chceme mít linku vykreslenou, vykreslí se jako neviditelná. Barvu jinak specifikujeme již známým způsobem. Volitelně si můžeme nechat umístit do legendy řetězec s popisem, co daná linka znamená. Hodnoty linky mohou být generovány pomocí DEF, VDEF a CDEF. V případě použití volitelného parametru `STACK` bude tato linka umístěna nad předchozí element, kterým může být právě jiná linka (`LINE`) nebo oblast (`AREA`).

Funkce pro vykreslování grafů běžně zajišťuje, aby celá oblast linky nebo oblasti byla v grafu viditelná. Měřítko grafu může být v případě nutnosti upraveno. Jakákoliv linka nebo oblast může být vyřazena z tohoto procesu přidáním volby `skipscale`.

Pomocí volby `dashes` můžeme upravovat vzhled vykreslovaných linek. V defaultním nastavení a při povolení této volby se vykreslí symetrická čárkovaná čára o délce jedné čárky 5 pixelů. Dále si zde můžeme definovat vlastní styl čárkovaných a čerchovaných čar, kdy každá zadaná číselná hodnota v pixelech určuje délku vypnutých/zapnutých pixelů.

Parametr `dash-offset` specifikuje odsazení šablony od počátku vykreslování.

`AREA:``value[#color][:[legend][:STACK][:skipscale]]`

`AREA` je obdobná jako `LINE`. Jediným rozdílem je, že u oblasti je vybarvená oblast mezi osou X a vykreslovanou linkou.

`SHIFT:``vname:offset`

Pomocí tohoto parametru může RRDtool vykreslit následující elementy se zvoleným časovým posunem. Chceme-li se, například, podívat zpět obden, zvolíme ofset v sekundách 60*60*24 = 86 400. Časový posun můžeme zadat číslem nebo proměnnou.

`TEXTALIGN:``{left|right|justified|center}`

Popisky jsou umísťovány pod graf. Když „přetečou“ na jednom řádku, zalomí se a začnou se zobrazovat na dalším. V defaultním nastavení se dále zarovnávají napravo a nalevo. Pomocí příkazu `TEXTALIGN` můžeme specifikovat, jak se mají zarovnávat.

Kapitoly A.3 - A.9 byly převzaty a přeloženy z oficiální anglické dokumentace nástroje RRDtool, která je umístěná na jeho domovských stránkách [16].

B Ukázkový skript pro sběr informací napsaný v jazyce Perl

```
#!/usr/bin/perl
use RRDs;

my $start=time;
my $rrd="/home/ubuntu/Documents/rrd/voltage/fanSpeeds.rrd";

my $graphSmall1h="/home/ubuntu/Documents/rrd/voltage/fanSpeedsSmall1h.svg";
my $graphLarge1h="/home/ubuntu/Documents/rrd/voltage/fanSpeedsLarge1h.svg";
my $graphSmall6h="/home/ubuntu/Documents/rrd/voltage/fanSpeedsSmall6h.svg";
my $graphLarge6h="/home/ubuntu/Documents/rrd/voltage/fanSpeedsLarge6h.svg";
my $graphSmall12h="/home/ubuntu/Documents/rrd/voltage/fanSpeedsSmall12h.svg";
my $graphLarge12h="/home/ubuntu/Documents/rrd/voltage/fanSpeedsLarge12h.svg";

 if  (not  -f $rrd)  {
RRDs::create ($rrd,"--start",$start-1,"--step",60,
"DS:fanCpu:GAUGE:120:600:7200",
"DS:fanCase:GAUGE:120:600:7200",
"DS:fanPower:GAUGE:120:600:7200",
"RRA:AVERAGE:0.5:1:1560",
);
my $ERROR=RRDs::error;
die "$0: unable to create '$rrd': $ERROR\n" if $ERROR;
};

$fanCpu=(`sensors -u | awk '/fan1_input/ {print substr(\$2,1,4)}' `) ;
chomp($fanCpu);
$fanCase=(`sensors -u | awk '/fan2_input/ {print substr(\$2,1,4)}' `) ;
chomp($fanCase);
$fanPower=(`sensors -u | awk '/fan3_input/ {print substr(\$2,1,4)}' `) ;
chomp($fanPower);

#print  $fanCpu;
#print  $fanCase;
#print  $fanPower;

RRDs::update $rrd,"$start:$fanCpu:$fanCase:$fanPower";

### GRAPH DEFINITION ###
sub printGraph{
RRDs::graph
@_,
"--end=now",
"--title=Monitorovani rychlosti otacek ventilatoru",
"--vertical-label=Otacky za minutu [RPM]",
"--watermark=Created by David Jucha",
"--imgformat=SVG",
### OTHER ###
"--rigid",
"--upper-limit=2700",
"--lower-limit=1000",
"--grid-dash=2:1",
```

```
”−−dynamic−labels”,
”−−slope−mode”,
”−−units−exponent=0”,
### COLORS ###
”−−color=CANVAS#2E2E1F”,
”−−color=FONT#4C4C33”,
”−−color=AXIS#990000”,
”−−color=ARROW#990000”,
### FONTS ###
”−−font=AXIS:9”,
”−−font=UNIT:12”,
”−−font=TITLE:14”,
”−−font=LEGEND:8”,
”−−font=WATERMARK:7”,

### DATA DEFINITION ###
”DEF:fanCpu=$rrd:fanCpu:AVERAGE”,
”VDEF:minFanCpu=fanCpu,MINIMUM”,
”VDEF:maxFanCpu=fanCpu,MAXIMUM”,
”VDEF:avgFanCpu=fanCpu,AVERAGE”,

”DEF:fanCase=$rrd:fanCase:AVERAGE”,
”VDEF:minFanCase=fanCase,MINIMUM”,
”VDEF:maxFanCase=fanCase,MAXIMUM”,
”VDEF:avgFanCase=fanCase,AVERAGE”,

”DEF:fanPower=$rrd:fanPower:AVERAGE”,
”VDEF:minFanPower=fanPower,MINIMUM”,
”VDEF:maxFanPower=fanPower,MAXIMUM”,
”VDEF:avgFanPower=fanPower,AVERAGE”,

### LINES AND LEGEND ###
”LINE2.5:fanCpu#CC0000:␣Ventilator␣na␣procesoru\\:␣␣␣␣”,
”LINE2.5:fanCase#333399:␣Ventilator␣v␣PC␣skrini\\:␣␣␣␣”,
”LINE2.5:fanPower#007A00:␣Ventilator␣ve␣zdroji␣napajeni\\:\\n”,

”LINE1.5:minFanCpu#85460000:␣Minimalni␣RPM\\:”,
”GPRINT:minFanCpu:\%6.0lf␣␣␣␣␣␣”,
”LINE1.5:minFanCase#85460000:␣Minimalni␣RPM\\:”,
”GPRINT:minFanCase:\%6.0lf␣␣␣␣␣”,
”LINE1.5:minFanPower#85460000:␣Minimalni␣RPM\\:”,
”GPRINT:minFanPower:\%6.0lf\\n”,

”LINE1.5:maxFanCpu#5599AA00:␣Maximalni␣RPM\\:”,
”GPRINT:maxFanCpu:\%6.0lf␣␣␣␣␣␣”,
”LINE1.5:maxFanCase#5599AA00:␣Maximalni␣RPM\\:”,
”GPRINT:maxFanCase:\%6.0lf␣␣␣␣␣”,
”LINE1.5:maxFanPower#5599AA00:␣Maximalni␣RPM\\:”,
”GPRINT:maxFanPower:\%6.0lf\\n”,

”LINE1.5:avgFanCpu#78952100:␣Prumerne␣RPM\\:␣”,
”GPRINT:avgFanCpu:\%6.0lf␣␣␣␣␣␣”,
”LINE1.5:avgFanCase#78952100:␣Prumerne␣RPM\\:␣”,
”GPRINT:avgFanCase:\%6.0lf␣␣␣␣␣”,
```

```
”LINE1.5:avgFanPower#78952100:␣Prumerne␣RPM\\:␣”,
”GPRINT:avgFanPower:\%6.0lf\\n”
;
}

### GRAPHS ###
printGraph(
”$graphSmall1h”,
”−−start=now−1h”,
”−−width=640”,
”−−height=360”,
”−−x−grid=MINUTE:1:MINUTE:5:MINUTE:5:0:%R”,
);

printGraph(
”$graphLarge1h”,
”−−start=now−1h”,
”−−width=800”,
”−−height=600”,
”−−x−grid=MINUTE:1:MINUTE:5:MINUTE:5:0:%R”,
);

printGraph(
”$graphSmall6h”,
”−−start=now−6h”,
”−−width=640”,
”−−height=360”,
”−−x−grid=MINUTE:10:HOUR:1:HOUR:1:0:%R”,
);

printGraph(
”$graphLarge6h”,
”−−start=now−6h”,
”−−width=800”,
”−−height=600”,
”−−x−grid=MINUTE:10:HOUR:1:HOUR:1:0:%R”,
);

printGraph(
”$graphSmall12h”,
”−−start=now−12h”,
”−−width=640”,
”−−height=360”,
”−−x−grid=MINUTE:30:HOUR:1:HOUR:1:0:%R”,
);

printGraph(
”$graphLarge12h”,
”−−start=now−12h”,
”−−width=800”,
”−−height=600”,
”−−x−grid=MINUTE:30:HOUR:1:HOUR:1:0:%R”,
);
```

```
if ($ERROR=RRDs::error){
print "ERROR: $ERROR\n";
};
```

Výpis 14: Ukázkový skript pro sběr informací o otáčkách ventilátorů

C Sada měření nezařazených do hlavní části práce

C.1 Lokální sběr informací z PC, notebooku nebo serveru

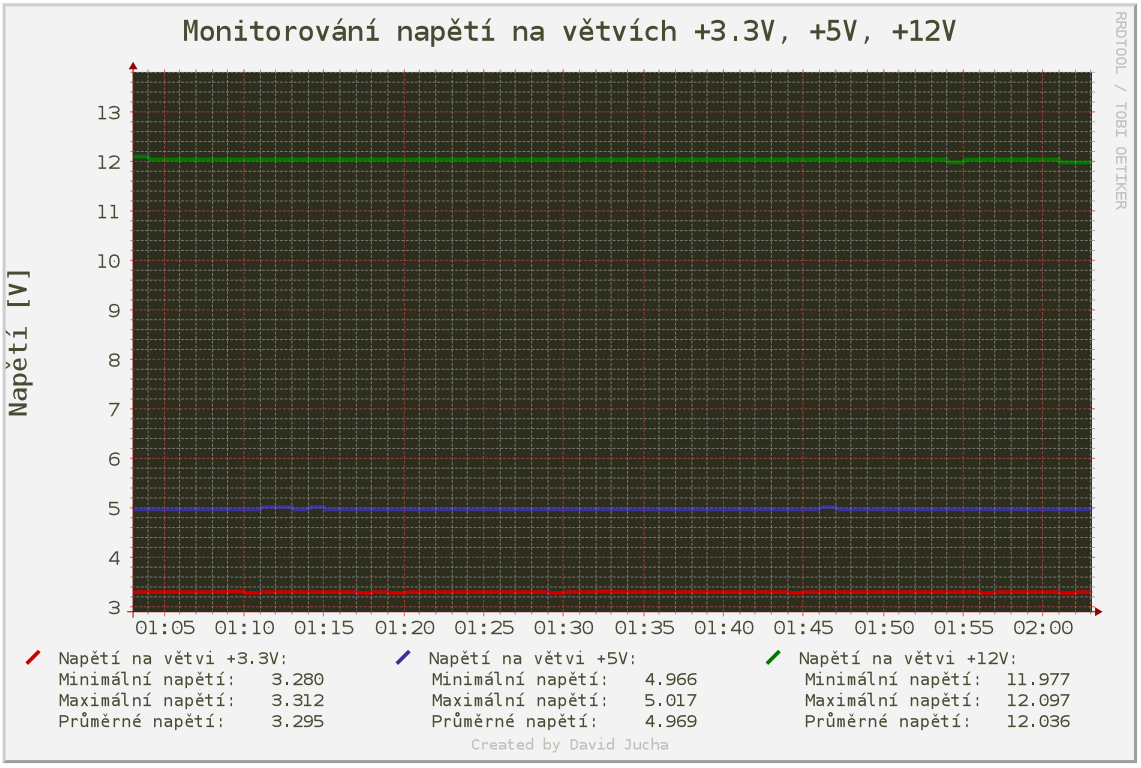
C.1.1 Napětí na jednotlivých větvích zdroje (+3,3 V; +5 V; +12 V)

Obdobně jako u napětí na procesoru můžeme monitorovat i jiná napětí. Jedná se o napětí na jednotlivých větvích napájecího zdroje počítače (obrázek C.1). Tyto větve odpovídají napětím 3,3 V; 5 V a 12 V. Jednotlivé větve může být potřeba monitorovat pro zajištění stability komponent, které tyto větve napájejí. Sesbírané informace nám také mohou pomoci při hledání příčin vypovězení jednotlivých komponent dané stanice.

Při vytváření příkazu pro sběr informací se postupuje obdobným způsobem jako u napětí pro procesor. První změna je v počtu příkazů, jelikož pro zobrazování trojích hodnot potřebujeme logicky tři příkazy. Druhá změna je v typech hodnot, které zde obsahují řetězec `in1_input`, `in2_input` a `in3_input`.

Výsledné příkazy potom mají následující podobu:

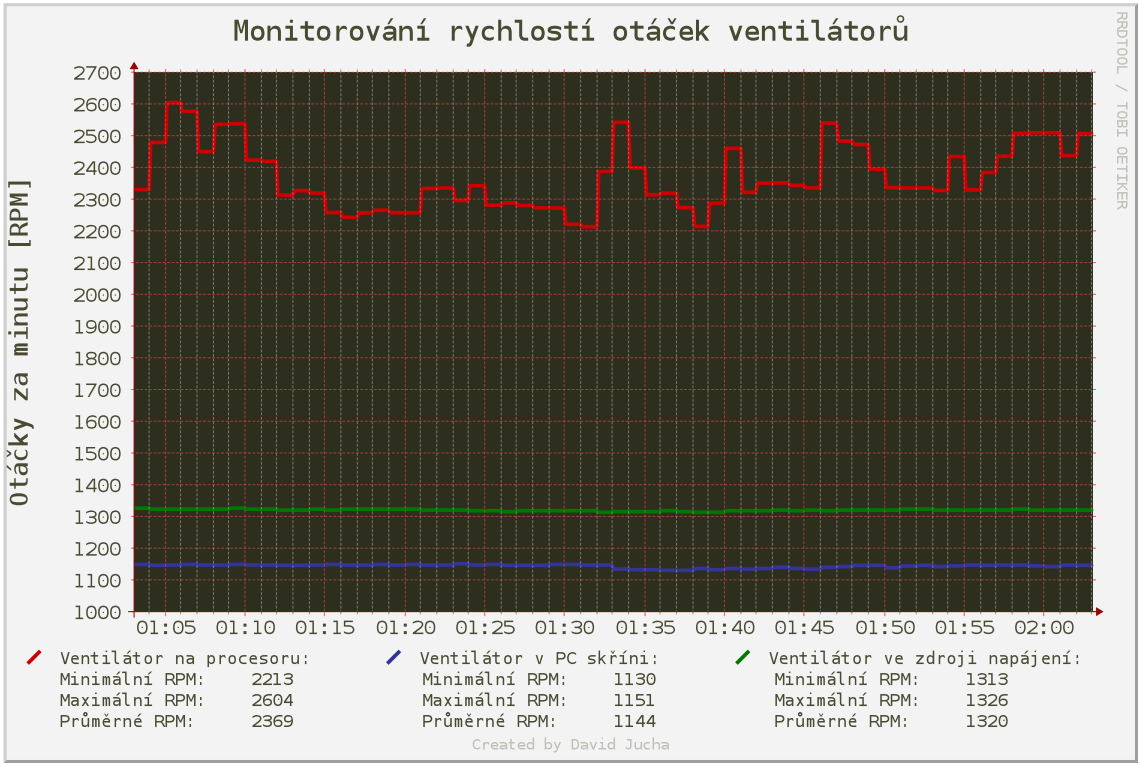
```
# sensors -u | awk '/in1_input/ {print $2}'
# sensors -u | awk '/in2_input/ {print $2}'
# sensors -u | awk '/in3_input/ {print $2}'
```



Obrázek C.1: Napětí na jednotlivých větvích za poslední hodinu

C.1.2 Rychlosti otáček ventilátorů

Dalšími zajímavými informacemi, které by mohly administrátory zajímat, jsou rychlosti otáček jednotlivých ventilátorů, umístěných v počítači (obrázek C.2). Jedná se o ventilátor ve zdroji napájení, na procesoru a ve skříni počítače. Tyto informace mohou být užitečné pro zjištění správné funkcionality ventilátorů nebo potenciometrů, pomocí kterých můžeme tyto ventilátory ovládat. Také můžeme sledovat, za jak dlouho se po větším vytížení a tím zahřátí počítače, vrátí otáčky do standardních hodnot.



Obrázek C.2: Rychlosti otáček ventilátorů za poslední hodinu

Pro získání informací o rychlostech ventilátorů se zprvu postupuje obdobně, jako u předchozího měření (zobrazují se taky 3 měřené hodnoty). Změna nastává opět v typech hodnot, zde se jedná o řetězce `fan1_input`, `fan2_input` a `fan3_input`. Kdybychom vše nechali již beze změn, budeme dostávat hodnotu ve tvaru: `XXXX.XXX`. To my ale u otáček nepotřebujeme, jelikož se hodnoty mění pouze v celých číslech. Jednoduchým řešením je přidání funkce `substr` do příkazu `awk`. Tato funkce, jak již sám název napovídá, umí vytvořit z daného řetězce podřetězec. Jedním z povolených formátů této funkce je `substr(x,y,z)`, kde `x` je zdrojový řetězec, `y` je pořadí znaku, od kterého se má podřetězec začít vybírat a `z` je počet znaků, které se mají vybrat. V našem případě potřebujeme jen první 4 znaky z řetězce `$2`, tzn. že výsledný tvar příkazu `substr` bude `substr($2,1,4)`.

Výsledné příkazy pro získávání hodnot o rychlostech otáček ventilátorů pak budou:

```
# sensors -u | awk '/fan1_input/ {print substr($2,1,4)}'
# sensors -u | awk '/fan2_input/ {print substr($2,1,4)}'
# sensors -u | awk '/fan3_input/ {print substr($2,1,4)}'
```

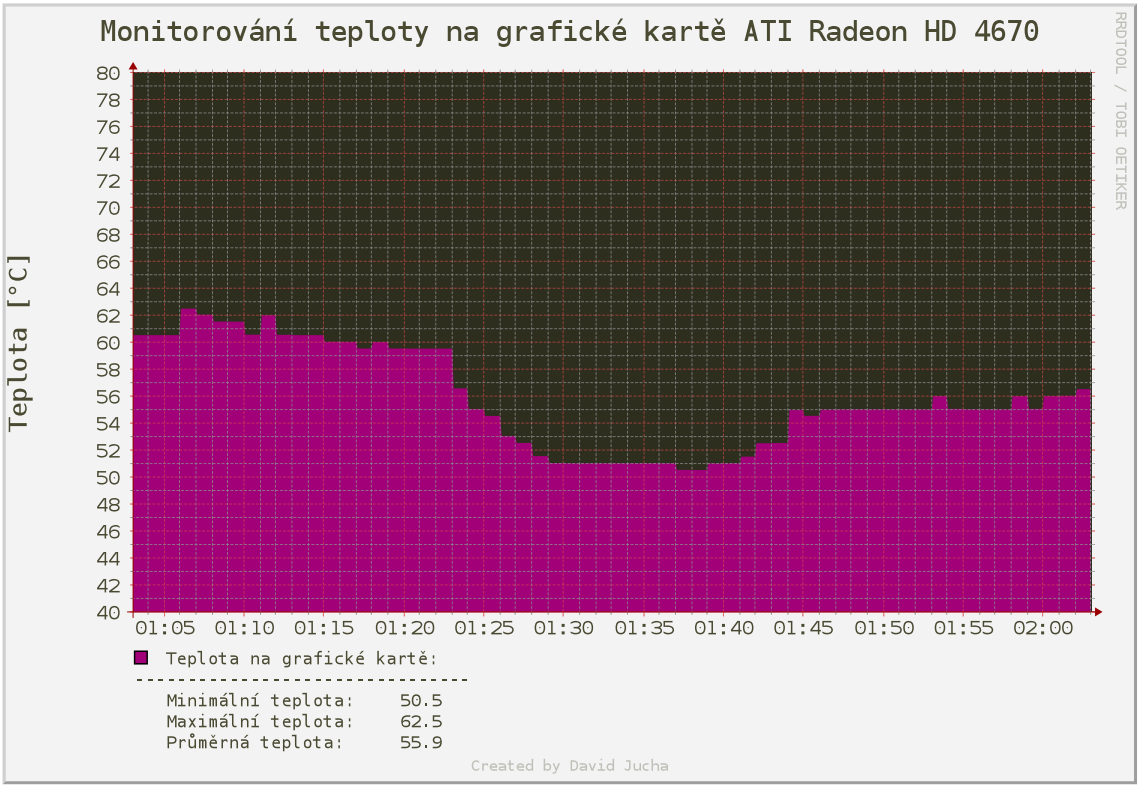
C.1.3 Teplota na grafické kartě ATI Radeon HD4670

Také teplotu na grafické kartě můžeme pomocí příkazu `# sensors` monitorovat (obrázek C.3). Naměřené hodnoty by mohly zajímat například herní nadšence, kteří zatěžují svou herní sestavu posledními vydanými tituly známých her. Nemusí však jít pouze o tuto skupinu lidí, složité výpočty na grafických kartách se používají i na profesionální úrovni ve světě vědy či 3D grafiky. I zde by nás tedy měly zajímat teploty na grafickém čipu.

Příkaz pro získávání informací o teplotě grafické karty je totožný s příkazem pro získávání teploty procesoru. Jediný rozdíl je v tom, že nevybíráme druhý řádek, ale ten první.

Příkaz má potom tvar:

```
# sensors -u | awk '/temp1_input/ {print substr($2,1,4)}'
| awk 'NR == 1'
```



Obrázek C.3: Teplota na grafické kartě za poslední hodinu

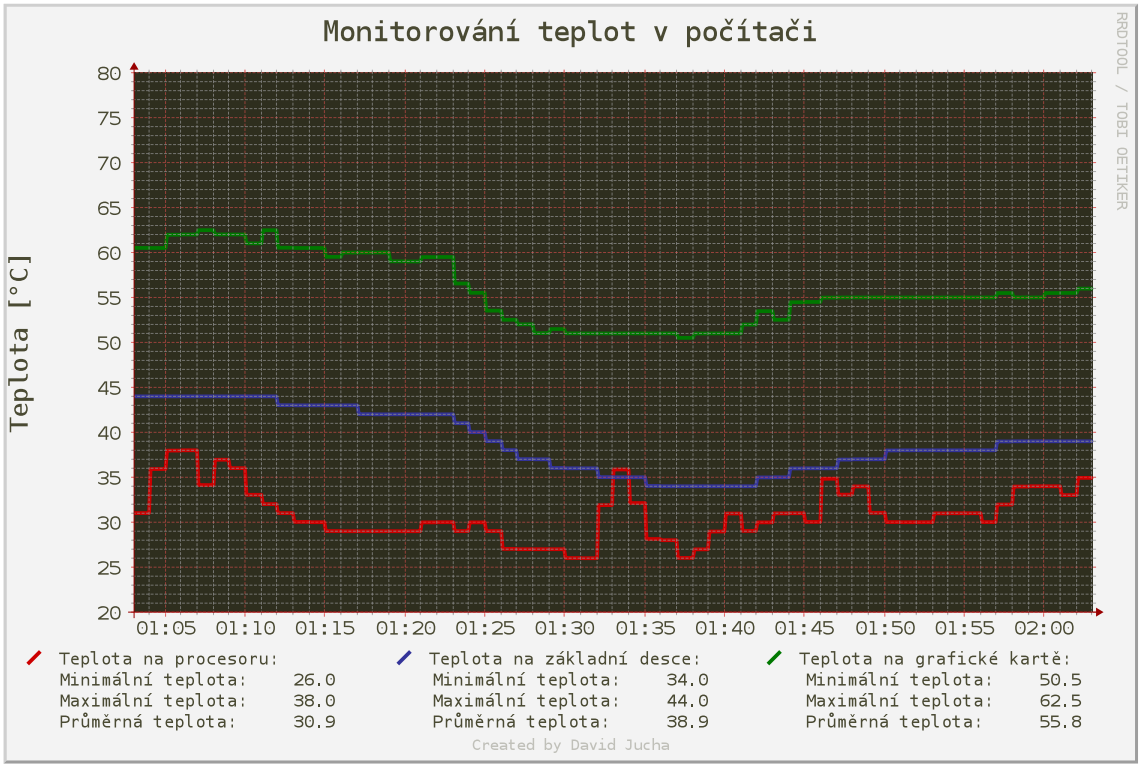
C.1.4 Teploty na základní desce, procesoru a grafické kartě

V posledním měření s teplotami jsem dal dohromady tři souhrnné teploty - teplotu na základní desce, procesoru a grafické kartě (obrázek C.4). Tyto teploty nám poskytují základní informace o teplotách v celé stanici. Můžeme zde porovnávat, jak jsou na sebe vázány, nebo to, které v případech jejich zvýšení klesají nejrychleji.

Získání informací o teplotě na procesoru a na grafické kartě je totožné s výše uvedenými měřeními. Teplota základní desky má řetězec, obsahující `typ_hodnoty` ve tvaru: `temp2_input` a vybíráme zde druhý řádek, jelikož jsou ve výpisu příkazu `# sensors` opět dvě hodnoty obsahující řetězec `temp2_input`.

Výsledné příkazy pak mají podobu:

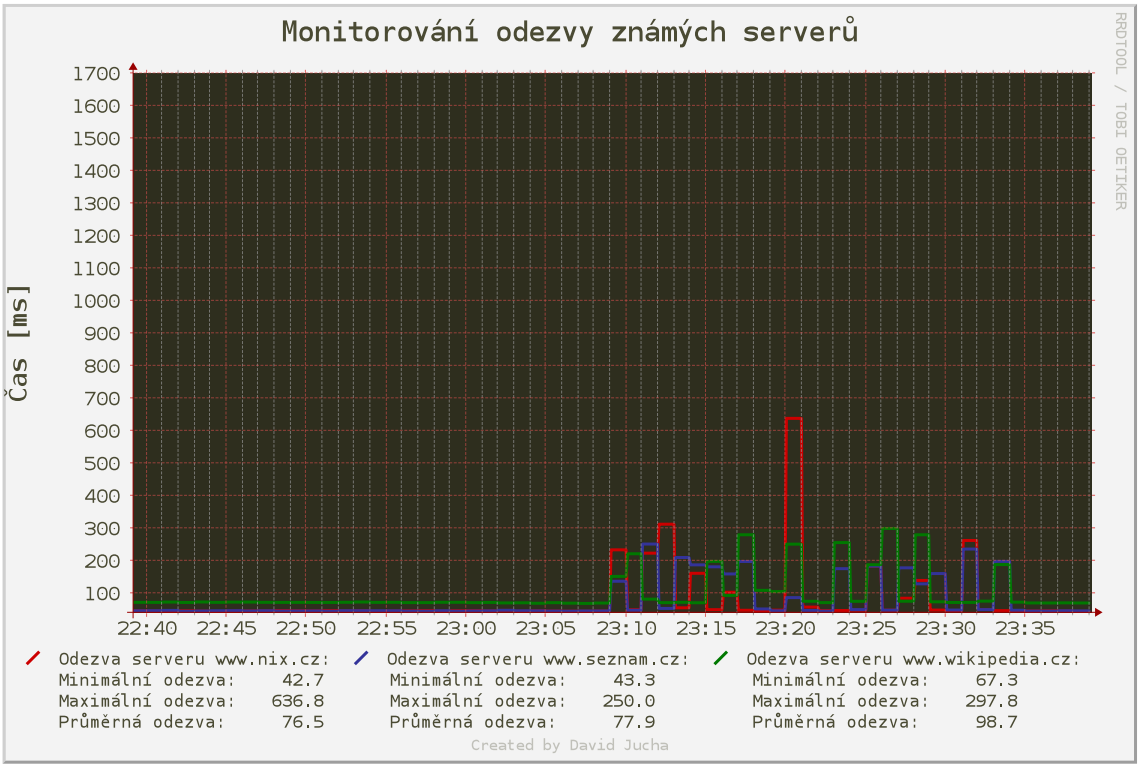
```
# sensors -u | awk '/temp1_input/ {print substr($2,1,4)}'
| awk 'NR == 2'

# sensors -u | awk '/temp1_input/ {print substr($2,1,4)}'
| awk 'NR == 1'

# sensors -u | awk '/temp2_input/ {print substr($2,1,4)}'
| awk 'NR == 1'
```



Obrázek C.4: Teploty na základní desce, procesoru a grafické kartě za poslední hodinu

C.1.5 Odezva známých serverů

Pro síťové administrátory může někdy nastat potřeba monitorovat z různých důvodů odezvu serverů. Může se jednat buď o veřejné servery, umístěné na veřejné síti - internetu, nebo jen o odezvu stanic, umístěných v jejich lokální síti. Já si pro ukázku zvolil servery z internetu, u stanic umístěných v lokální síti lze odezvu monitorovat obdobným způsobem. Pro mé potřeby jsem využil servery: `www.nix.cz`, `www.seznam.cz` a `www.wikipedia.cz`, běžící na IPv6 (obrázek C.5).



Obrázek C.5: Odezva známých serverů za poslední hodinu

K získání informací o odezvě stanic, běžících na IPv6, slouží v OS Ubuntu příkaz `ping6`. Ten jsem použil spolu s parametrem `-c 3`, který udává počet dotazů na odezvu, s parametrem `-q`, který nám vytiskne na výstup zkrácenou statistiku, ze které budeme následně čerpat informace a s parametrem, udávajícím IP adresu nebo URL (Uniform Resource Locator) požadovaných serverů. Z výpisu dále vybíráme pomocí příkazu `awk` řádek obsahující znaménko = a tiskneme na výstup podřetězec ze čtvrtého sloupce s minimální hodnotou odezvy, tj. prvních 5 znaků.

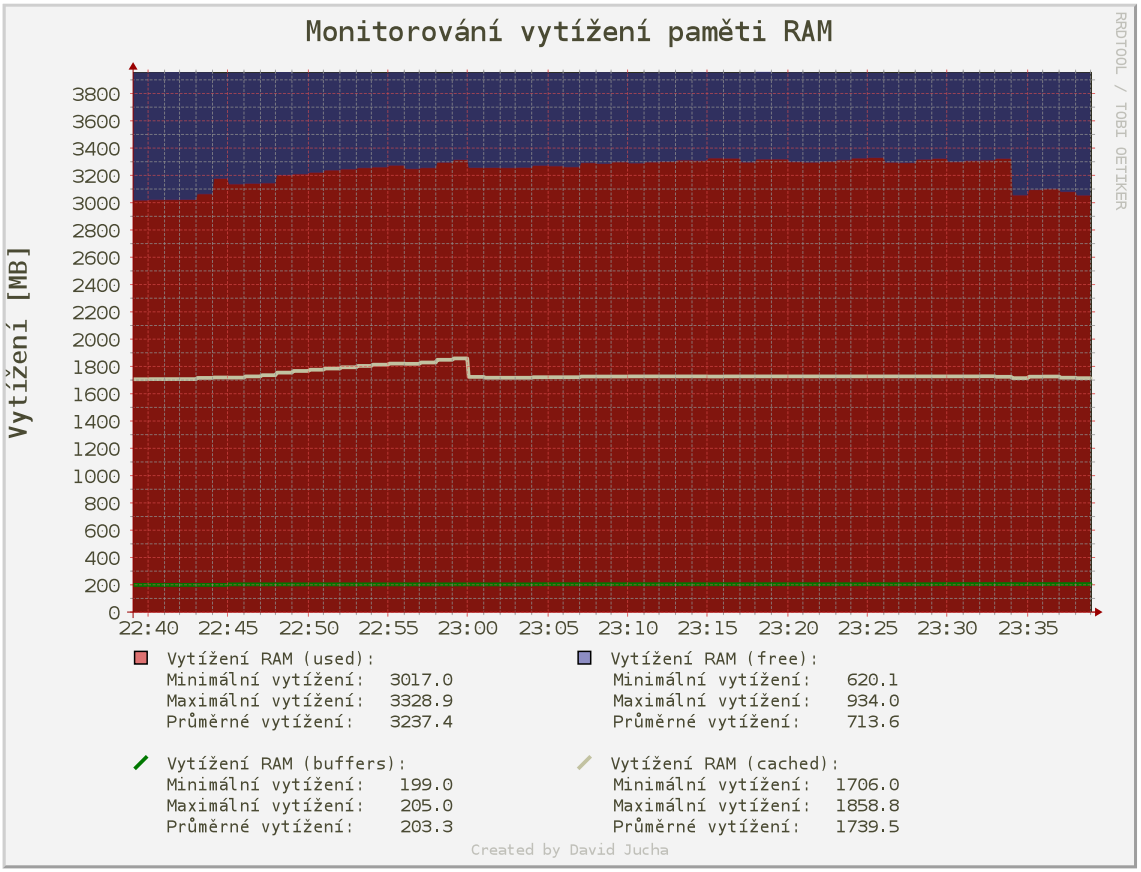
Výsledný příkaz pro získávání informací o odezvě má výsledný tvar:

```
# ping6 -c 3 -q nix.cz | awk '/=/ {print substr($4,1,5)}'
# ping6 -c 3 -q seznam.cz | awk '/=/ {print substr($4,1,5)}'
# ping6 -c 3 -q wikipedia.cz | awk '/=/ {print substr($4,1,5)}'
```

C.1.6 Vytížení paměti RAM

V dalším monitorování jsem se zaměřil na vytížení pamětí RAM (obrázek C.6). V dnešní době existuje mnoho řešení systémů, které běží pouze v pamětech RAM a především zde je přinejmenším vhodné monitorovat jejich vytížení. Získané informace nám mohou pomoci odhalit závady RAM pamětí nebo nás upozornit na to, že jejich kapacita je již nedostačující a je potřeba navýšit jejich kapacitu.

V OS Ubuntu slouží k vypsání základních informací o pamětech RAM příkaz `free`. Chceme získávat hodnoty v MB, proto použijeme parametr `-m`. Dále zřetězíme příkaz `free` příkazem `awk`, ve kterém vyhledáváme řádek, obsahující řetězec `Mem:`, a tiskneme hodnoty ze sloupců 3, 4, 6 a 7.



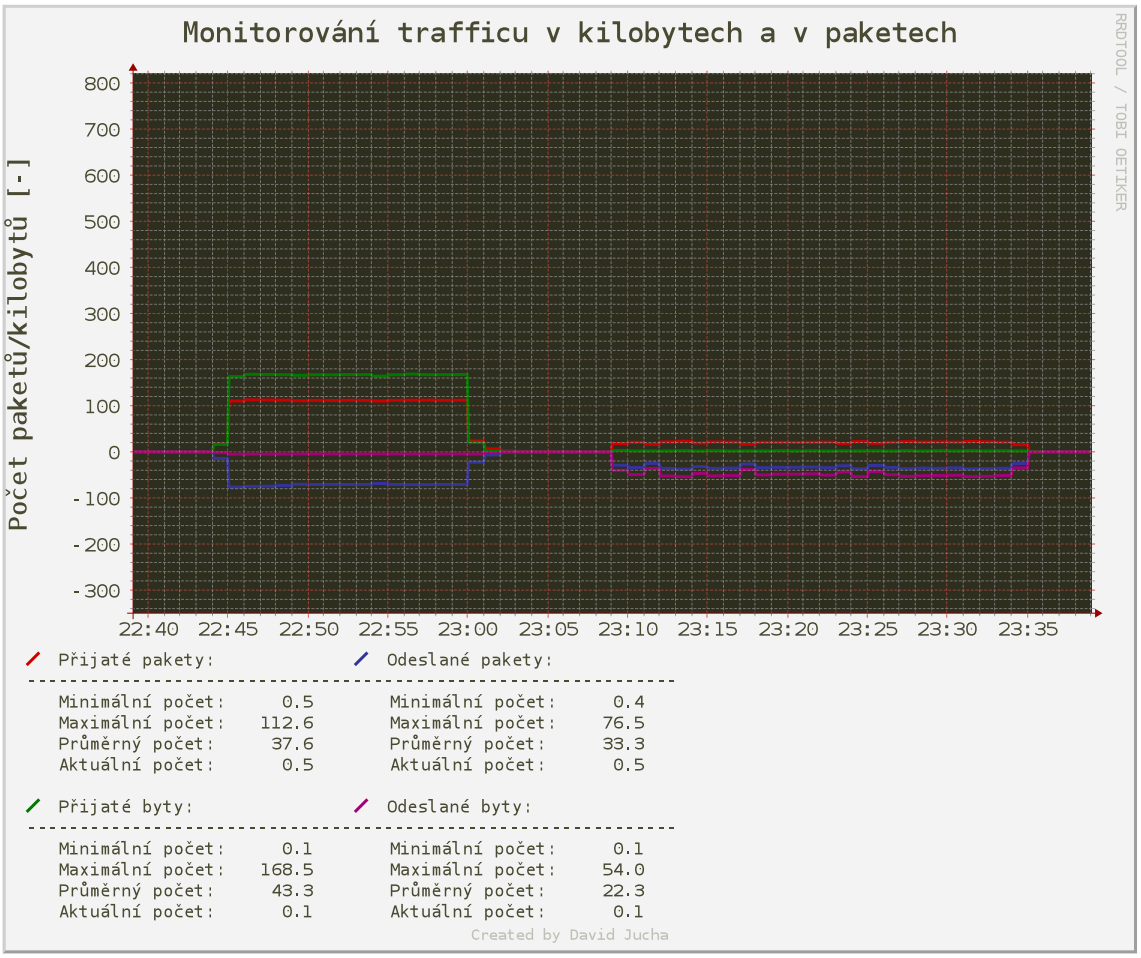
Obrázek C.6: Vytížení paměti RAM za poslední hodinu

Výsledné příkazy budou vypadat následovně:

```
# free -m | awk '/Mem:/ {print $3}'
# free -m | awk '/Mem:/ {print $4}'
# free -m | awk '/Mem:/ {print $6}'
# free -m | awk '/Mem:/ {print $7}'
```

C.1.7 Traffic v obou směrech v kilobytech a v paketech

Posledním měřením, které jsem se rozhodl umístit do podkapitoly s lokálními měřeními, je měření tzv. trafficu (obrázek C.7). Traffic se dá měřit různými způsoby v různých jednotkách a směrech. Rozhodl jsem se monitorovat oba směry, jak příchozí, tak i ten odchozí a jako jednotky jsem si zvolil pakety a kB. Tímto měřením mohou administrátoři zjišťovat zatížení jednotlivých rozhraní stanice, ať už v příchozím nebo odchozím směru a z výsledků vyvozovat různé závěry.



Obrázek C.7: Traffic v obou směrech v kilobytech a v paketech za poslední hodinu

Pro práci se síťovými rozhraními slouží v OS Ubuntu příkaz `ifconfig`, kterému jsem hned ze začátku přidal parametr `eth0`, definující rozhraní, které jsem chtěl monitorovat. Následně jsem jej zřetězil s příkazem `awk`, ve kterém jsem si nechal vyhledat řádek, obsahující řetězec „`RX packets:`“, ve kterém jsem pomocí funkce `sub` nahradil vyhledaný řetězec „`RX packets:`“ mezerou a vytiskl si obsah prvního sloupce. Pro ostatní potřebné hodnoty jsem provedl stejné operace, ale s tím rozdílem, že klíčový řetězec „`RX packets:`“ jsem nahrazoval ostatními potřebnými řetězci „`TX packets:`“,

„RX bytes:“ a „TX bytes:“. Následný přepočet z bytů na kilobyty jsem provedl opět až v Perlovských skriptech ve funkci Graph nástroje RRDtool.

Výsledné příkazy pro získávání informací ohledně trafficu mají tedy podobu:

```
ifconfig eth0 | awk '/RX packets:/ {sub(/RX packets:/,"");
print $1}'

ifconfig eth0 | awk '/TX packets:/ {sub(/TX packets:/,"");
print $1}'

ifconfig eth0 | awk '/RX bytes:/ {sub(/RX bytes:/,"");
print $1}'

ifconfig eth0 | awk '/TX bytes:/ {sub(/TX bytes:/,"");
print $5}'
```

C.2 Sběr informací z PC, notebooků nebo serverů v počítačové síti

C.2.1 Vytížení procesoru na stanicích DP2 a DP3

Stejně jako můžeme měřit vytížení procesoru lokálně pomocí vestavěných nástrojů OS Ubuntu, můžeme ho měřit i pomocí SNMP. U tohoto měření si myslím, že použití SNMP je lepší volba, než jeho nepoužití, jelikož provádění příkazu nemá takový vliv na vytížení počítače a tím i ovlivnění tohoto měření. V grafu na obrázku C.8 jsou najednou zobrazena jednotlivá vytížení (System, User, Nice) ze stanic DP2 a DP3. Všechny hodnoty potřebné pro měření se nacházejí v databázi UCD-SNMP-MIB, která odpovídá podstromu .1.3.6.1.4.1.2021. Přesné cesty pro získání hodnot o vytížení jsou následující:

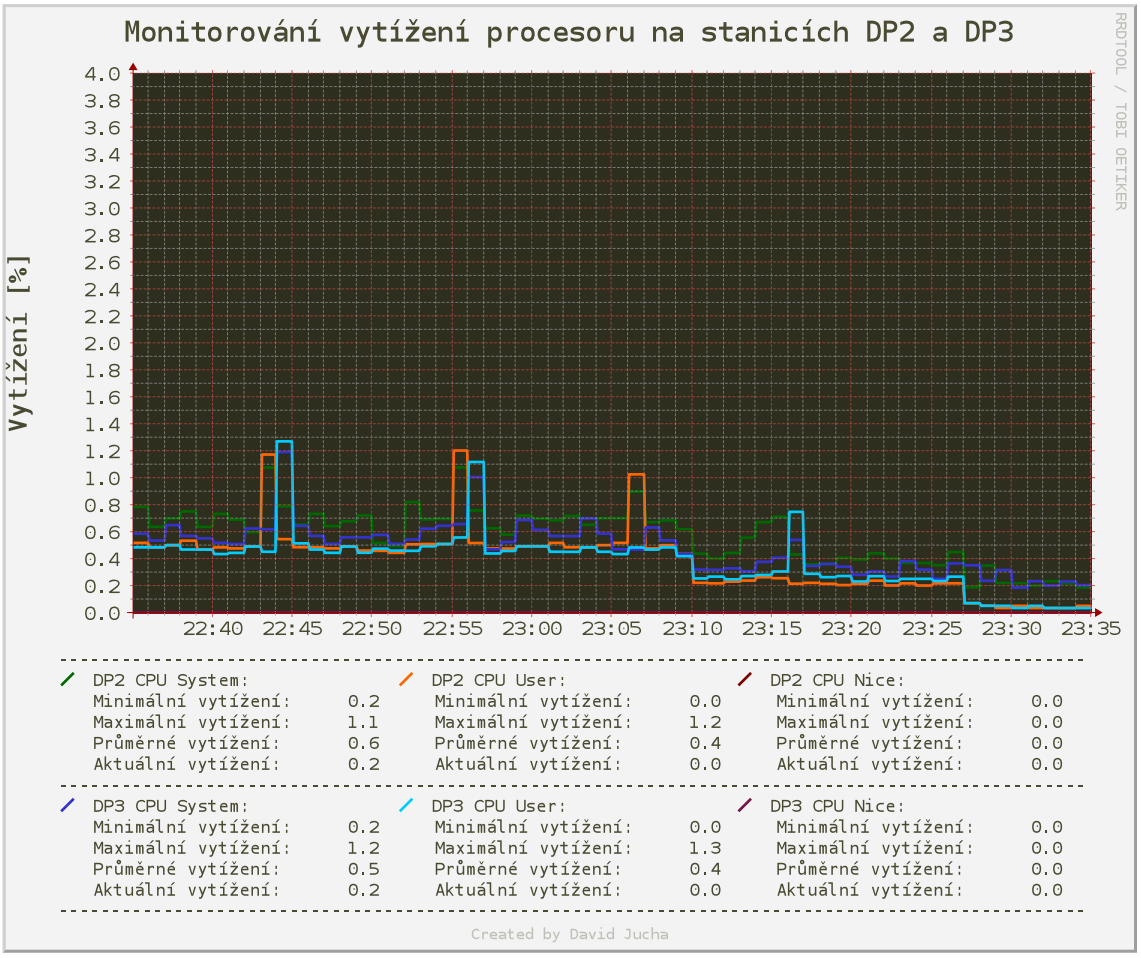
```
.1.3.6.1.4.1.2021.11.50.0 - ssCpuRawSystem.0
.1.3.6.1.4.1.2021.11.51.0 - ssCpuRawUser.0
.1.3.6.1.4.1.2021.11.52.0 - ssCpuRawNice.0
```

V případě, že máme na *manažerovi* nainstalován balíček snmp-mibs-downloader, můžeme při volání použít textové názvy uvedené za pomlčkami.

Celé příkazy pro získávaní informací mají tvar, zobrazený níže:

```
snmpget -v 2c -c public -m UCD-SNMP-MIB -Oqv
udp6:[2001:718:1001:2c6::182] ssCpuRawSystem.0

snmpget -v 2c -c public -m UCD-SNMP-MIB -Oqv
udp6:[2001:718:1001:2c6::182] ssCpuRawUser.0

snmpget -v 2c -c public -m UCD-SNMP-MIB -Oqv
udp6:[2001:718:1001:2c6::182] ssCpuRawNice.0

snmpget -v 2c -c public -m UCD-SNMP-MIB -Oqv
udp6:[2001:718:1001:2c6::183] ssCpuRawSystem.0

snmpget -v 2c -c public -m UCD-SNMP-MIB -Oqv
udp6:[2001:718:1001:2c6::183] ssCpuRawUser.0
```

```
snmpget -v 2c -c public -m UCD-SNMP-MIB -Oqv
udp6:[2001:718:1001:2c6::183] ssCpuRawNice.0
```

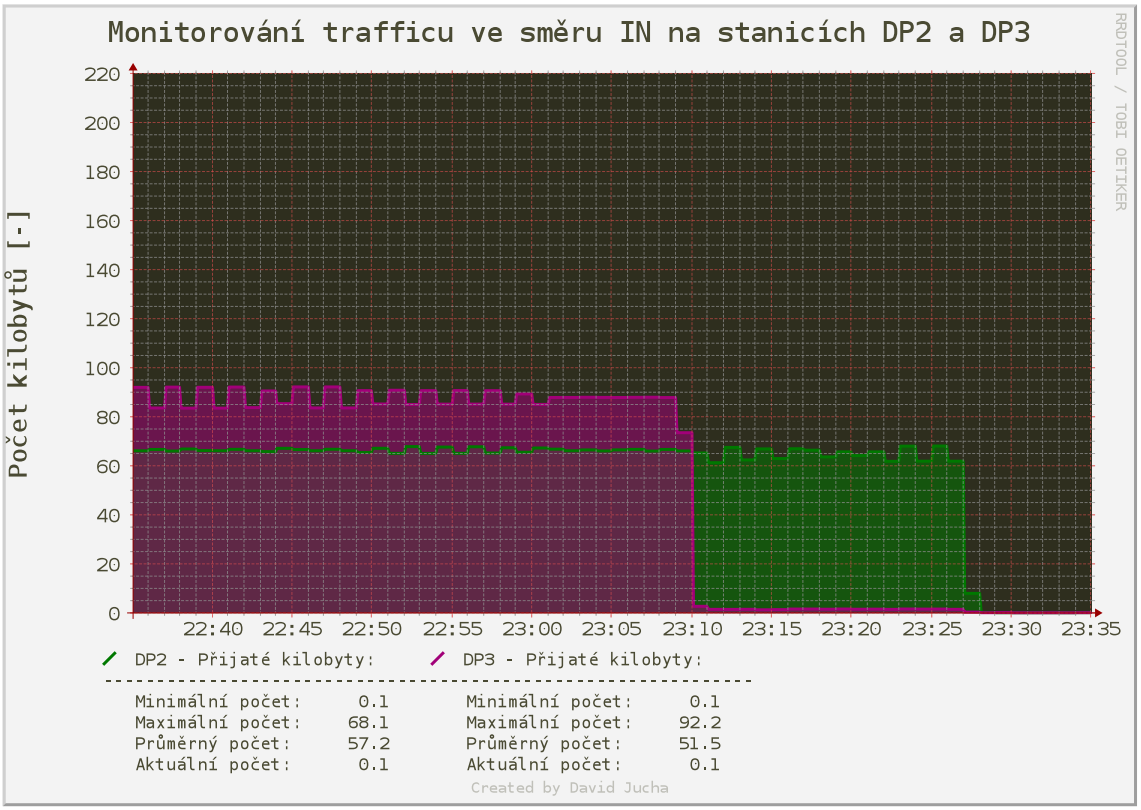


Obrázek C.8: Vytížení procesoru na stanicích DP2 a DP3 za poslední hodinu

C.2.2 Traffic ve směru IN na stanicích DP2 a DP3

Zajímavé a mnohdy potřebné je i monitorování provozu na počítačové síti - tzv. trafficu. V měření jsem se zaměřil vždy jen na jeden směr, aby byly grafy čitelnější a logicky oddělené. Směrem IN je myšlen download monitorovaných stanic DP2 a DP3. Informace o rozhraních jsou uloženy v databázi `IF-MIB`, které odpovídá OID `.1.3.6.1.2.1.2.2`. Při měření byly přenosové rychlosti uměle omezovány, aby v grafech vynikly naměřené hodnoty. Kompletní cesta pro získávání informací o provozu ve směru IN je:

```
1.3.6.1.2.1.2.2.1.10.2 - ifInOctets.2
```



Obrázek C.9: Traffic ve směru IN na stanicích DP2 a DP3 za poslední hodinu

Celé příkazy, pro získání hodnot s využitím SNMP protokolu jsou uvedeny zde:

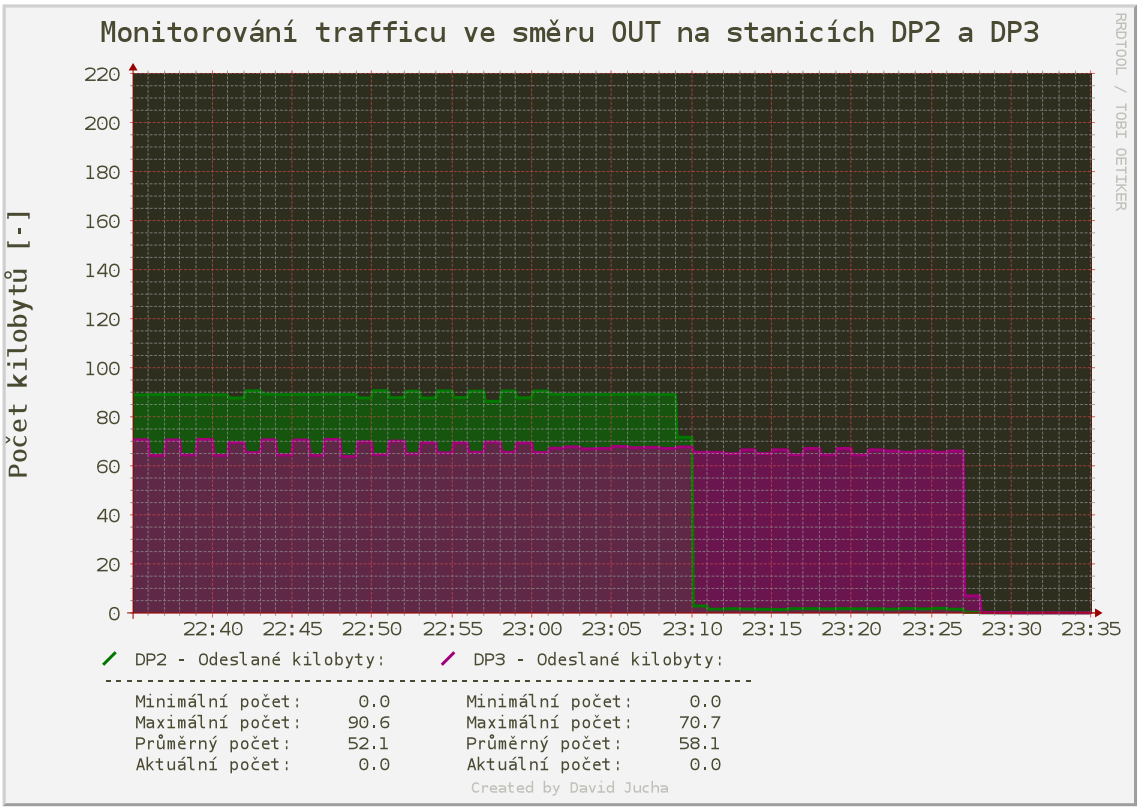
```
snmpget -v 2c -c public -m IF-MIB -Oqv
udp6:[2001:718:1001:2c6::182] ifInOctets.2
```

```
snmpget -v 2c -c public -m IF-MIB -Oqv
udp6:[2001:718:1001:2c6::183] ifInOctets.2
```

C.2.3 Traffic ve směru OUT na stanicích DP2 a DP3

Když už jsme u měření provozu, bylo by zarážející se nezmínit i o opačném směru provozu. Jedná se o směr OUT, tzv. upload. I zde jsou v grafu umístěny hodnoty z obou měřených stanic DP2 i DP3 (obrázek C.10). Hodnoty pocházejí ze stejných podstromů MIB databáze jako u předchozího měření.

Přenosové rychlosti byly uměle omezeny. Jelikož se jednalo o opačné směry oproti předchozímu měření a stanice jsem vytěžoval právě pomocí těchto dvou stanic, jsou si grafy odpovídající. Ta data, co se v předchozím měření ve stanici DP2 odesílají jsou přesně ta data, co se zde přijímají. Ten samý případ nastal i u druhé kombinace. Celá cesta v databázi MIB pro získávání hodnot pak je:



Obrázek C.10: Traffic ve směru OUT na stanicích DP2 a DP3 za poslední hodinu

```
1.3.6.1.2.1.2.2.1.16.2 - ifOutOctets.2
```

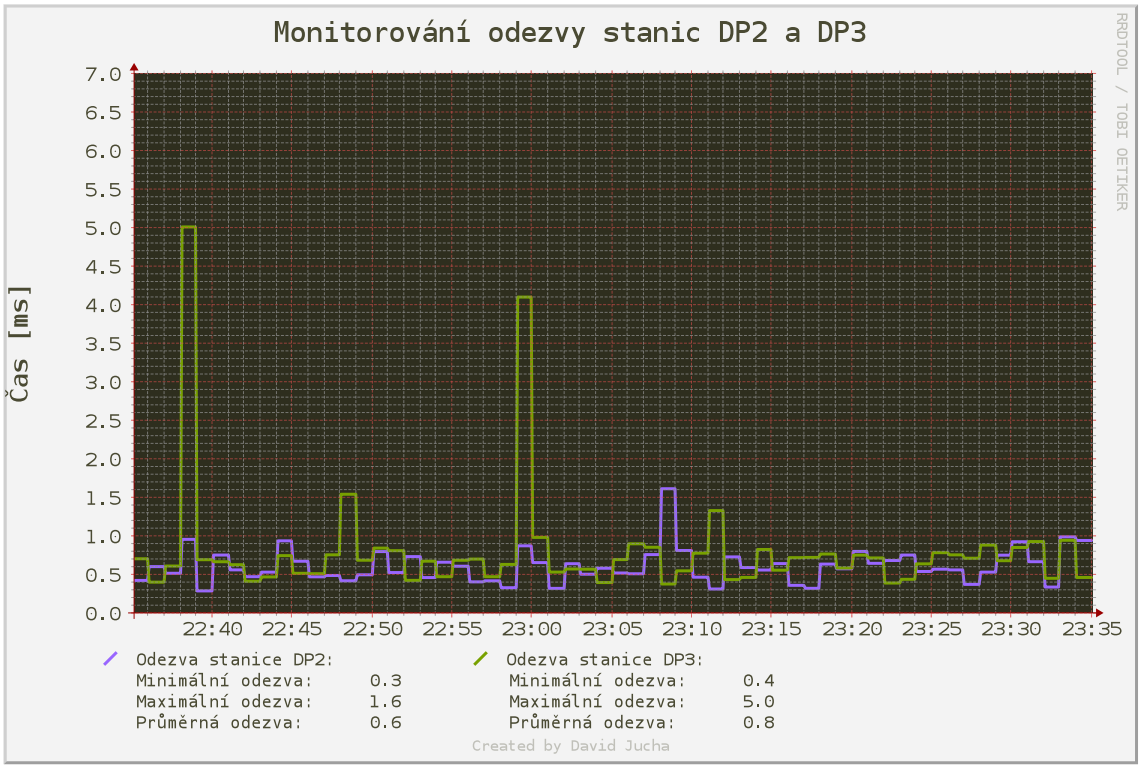
Celé příkazy pro získání hodnot pro směr OUT pak jsou:

```
snmpget -v 2c -c public -m IF-MIB -Oqv
udp6:[2001:718:1001:2c6::182] ifOutOctets.2
```

```
snmpget -v 2c -c public -m IF-MIB -Oqv
udp6:[2001:718:1001:2c6::183] ifOutOctets.2
```

C.2.4 Odezva stanic DP2 a DP3

Jako poslední měření v této podkapitole je měření odezvy stanic, zapojených v lokální síti. Jedná se odezvu, měřenou ze serveru DP1 směrem ke stanicím DP2 a DP3. Jelikož jsou všechny tyto stanice umístěny na jednom virtuálním serveru, jsou hodnoty odezvy velmi malé. Proto je graf přizpůsoben zobrazování těchto malých hodnot, kdy maximální hodnota odezvy je nastavena na 7 ms. U měření nebylo využito protokolu SNMP, ale klasického příkazu `ping6`, jelikož všechny tyto virtuální servery pracují na IPv6.



Obrázek C.11: Odezva stanic DP2 a DP3 za poslední hodinu

Celé příkazy mají následující tvar:

```
ping6 -c3 -q 2001:718:1001:2c6::182 | awk '/=/
{print substr($4,1,5)}'
```

```
ping6 -c3 -q 2001:718:1001:2c6::183 | awk '/=/
{print substr($4,1,5)}'
```

Detailní popis těchto příkazů naleznete v kapitole C.1.5, kde jsme se věnovali odezvě serverů, umístěných v internetu.

C.3 Sběr informací z jiných zařízení v počítačové síti

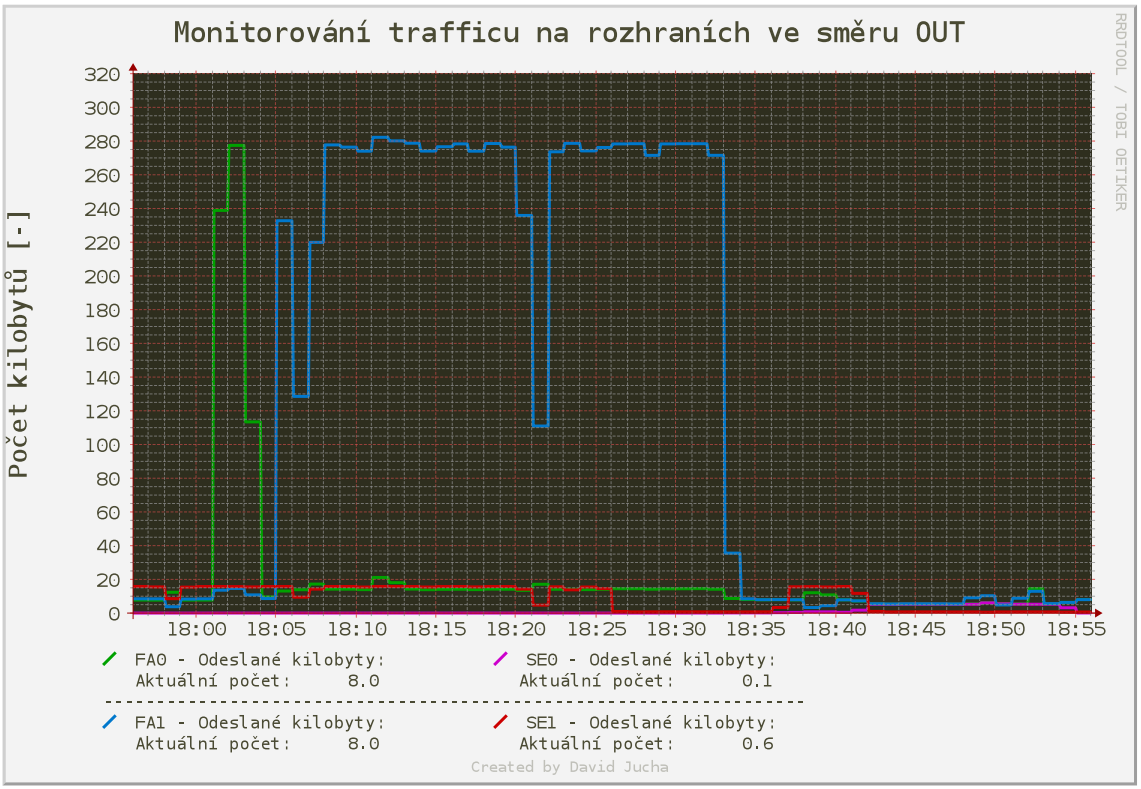
C.3.1 Traffic ve směru OUT na všech rozhraních routeru Cisco 2800

Podle stejného scénáře jako v podkapitole 4.3.3 probíhalo měření i v opačném směru provozu - tj. ve směru OUT neboli download. Informace o rozhraních jsou umístěny v databázi `IF-MIB`, té odpovídá OID `.1.3.6.1.2.1.2.2`.

Pro naše rozhraní a směr OUT měly příkazy pro získání námi potřebných informací následující tvar:

```
snmpwalk -v 2c -c public -Oqv 192.168.1.1
.1.3.6.1.2.1.2.2.1.16.1
```

```
snmpwalk -v 2c -c public -Oqv 192.168.1.1
.1.3.6.1.2.1.2.2.1.16.2
snmpwalk -v 2c -c public -Oqv 192.168.1.1
.1.3.6.1.2.1.2.2.1.16.3
snmpwalk -v 2c -c public -Oqv 192.168.1.1
.1.3.6.1.2.1.2.2.1.16.4
```



Obrázek C.12: Traffic ve směru OUT na všech rozhraních routeru Cisco 2800 za poslední hodinu

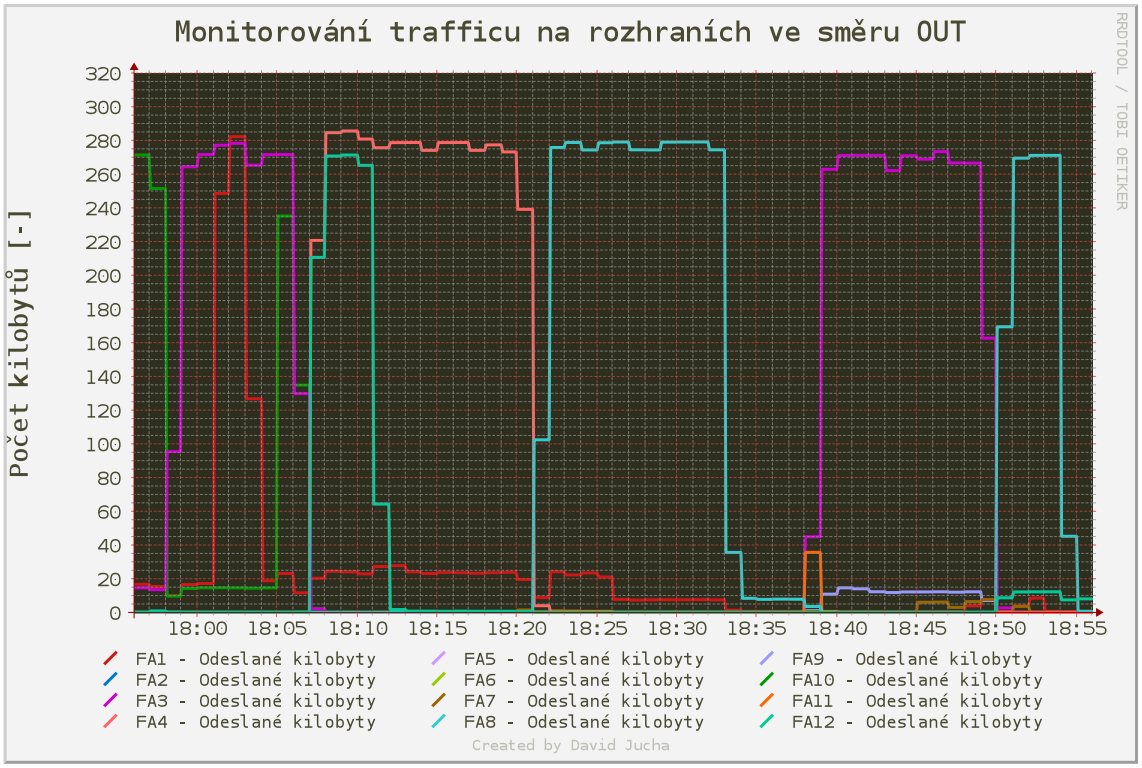
C.3.2 Traffic ve směru OUT na rozhraních switche Cisco Catalyst 3560

Obdobně jako jsem v kapitole 4.3.4 monitoroval provoz na prvních dvanácti portech ve směru IN, tak jsem zde monitoroval i směr opačný - směr OUT neboli upload. Naměřené hodnoty uploadu jsou umístěny v grafu (obrázek C.13). V měření pro oba směry byl provoz uměle omezován, aby byly grafy dobře čitelné. Reálné přenosové rychlosti jsou mnohonásobně vyšší.

Příkazy pro získávání informací, umísťovaných do grafů, vypadají následovně:

```
snmpwalk -v 2c -c public -Oqv 192.168.1.2
.1.3.6.1.2.1.2.2.1.16.10001
```

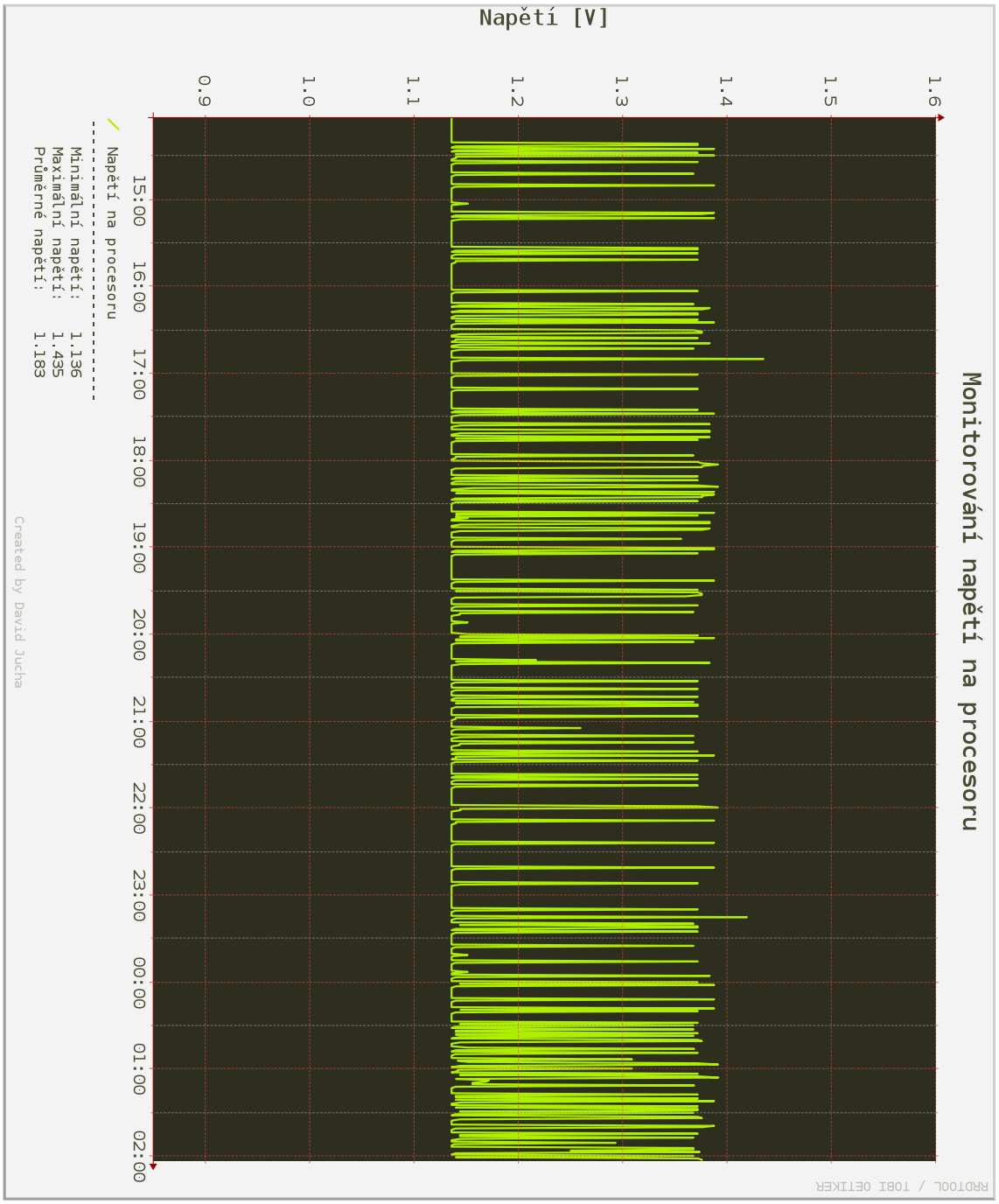
I zde se pro různé porty mění pouze poslední pětičíslí. Pro první port odpovídá hodnotě `10001`, pro mnou poslední monitorovaný pak hodnotě `10012`.



Obrázek C.13: Traffic ve směru OUT na dvanácti rozhraních switche Cisco Catalyst 3560 za poslední hodinu

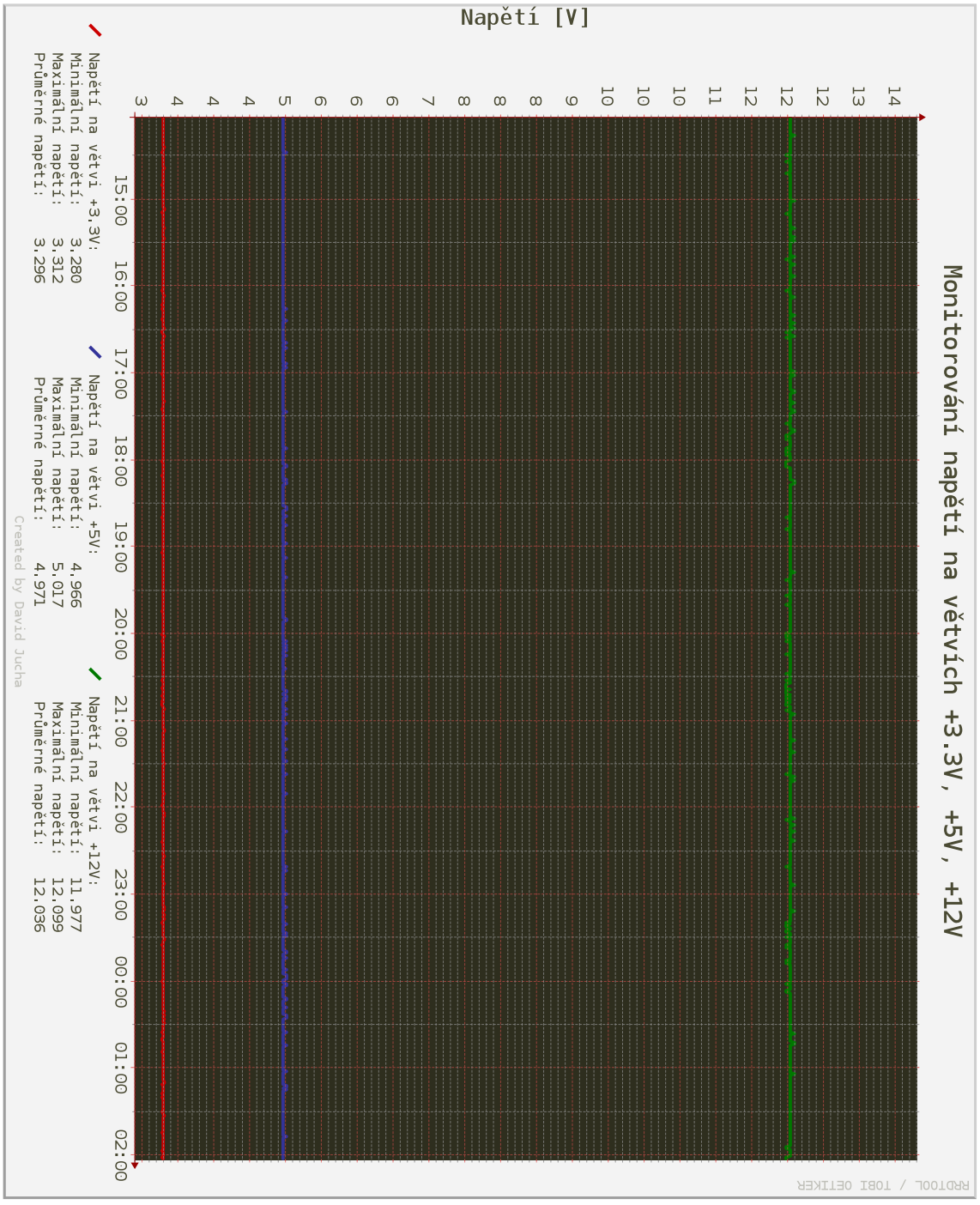
D Sada grafů z provedených měření

D.1 Sada grafů z lokální stanice za posledních 12 hodin



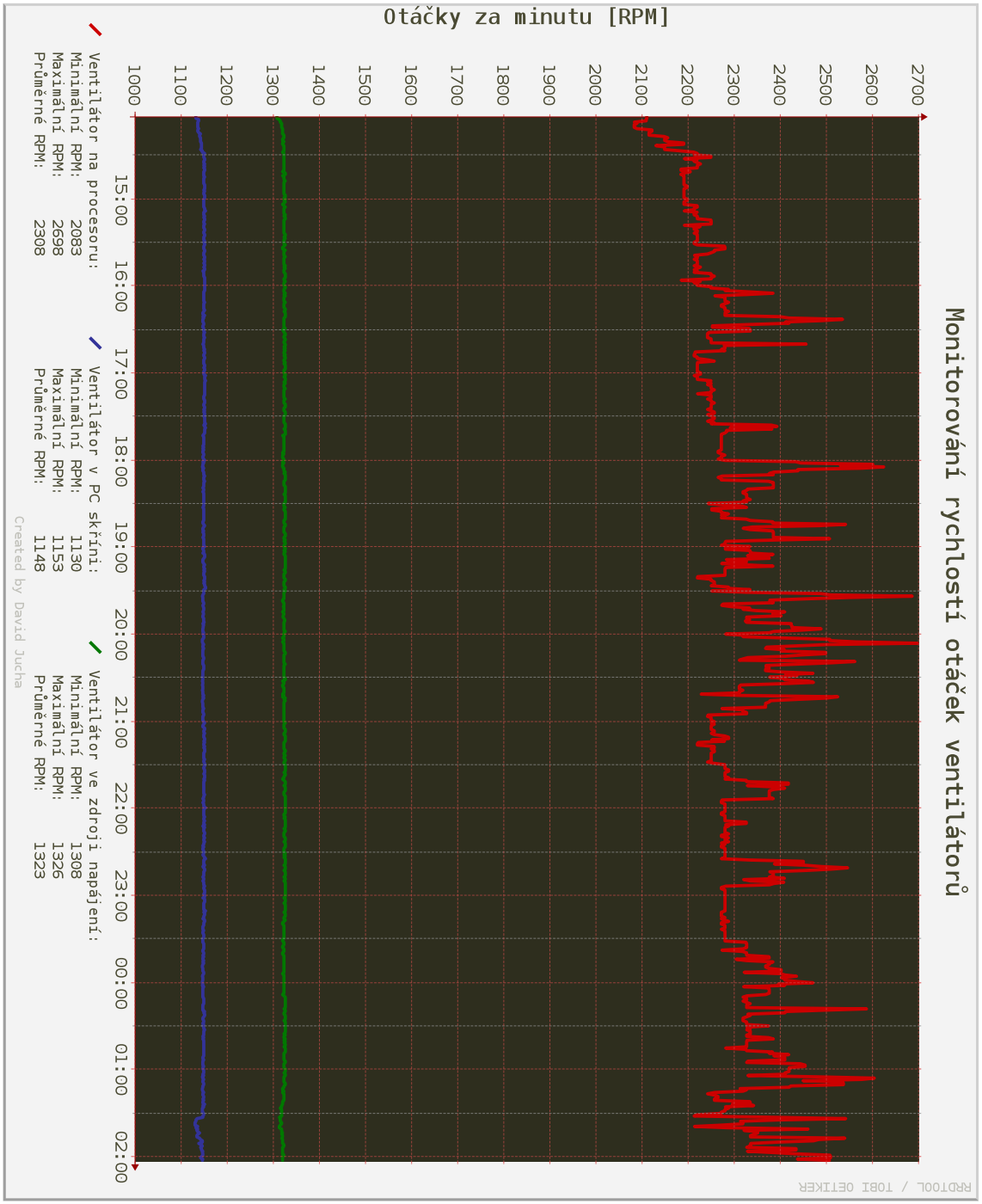
Obrázek D.1: Napětí na procesoru za posledních 12 hodin

D SADA GRAFŮ Z PROVEDENÝCH MĚŘENÍ



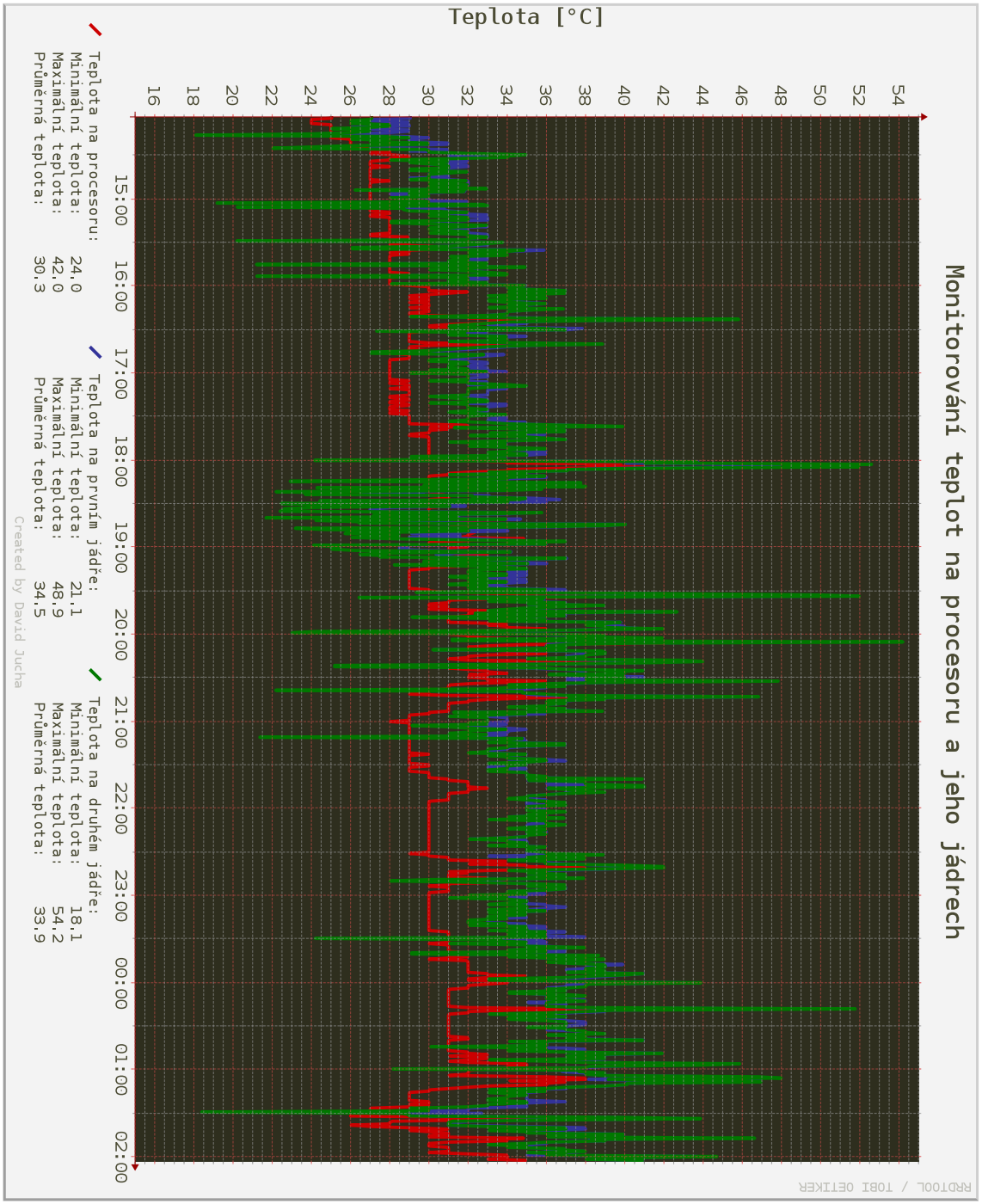
Obrázek D.2: Napětí na jednotlivých větvích za posledních 12 hodin

D SADA GRAFŮ Z PROVEDENÝCH MĚŘENÍ

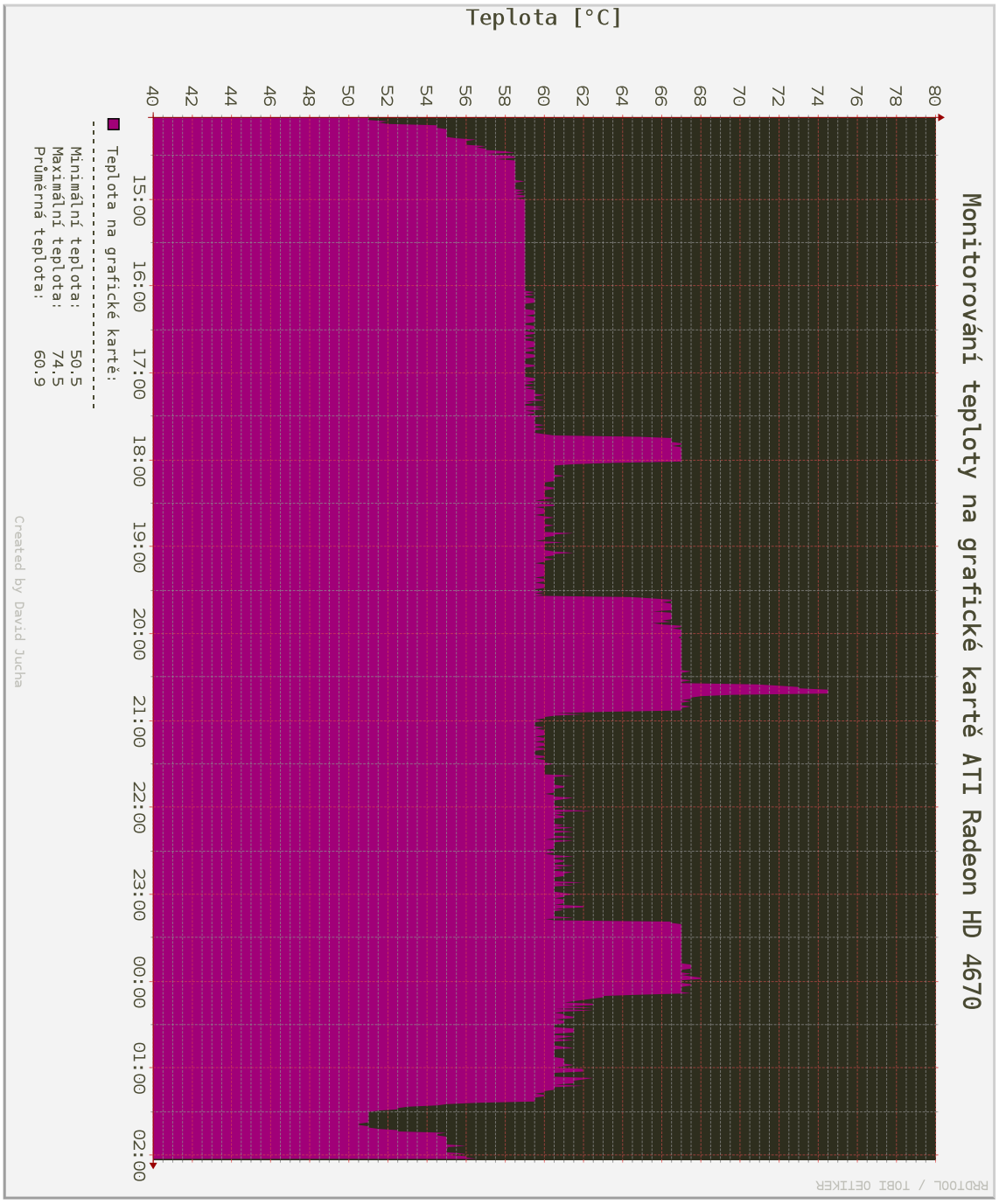


Obrázek D.3: Rychlosti otáček ventilátorů za posledních 12 hodin

D SADA GRAFŮ Z PROVEDENÝCH MĚŘENÍ

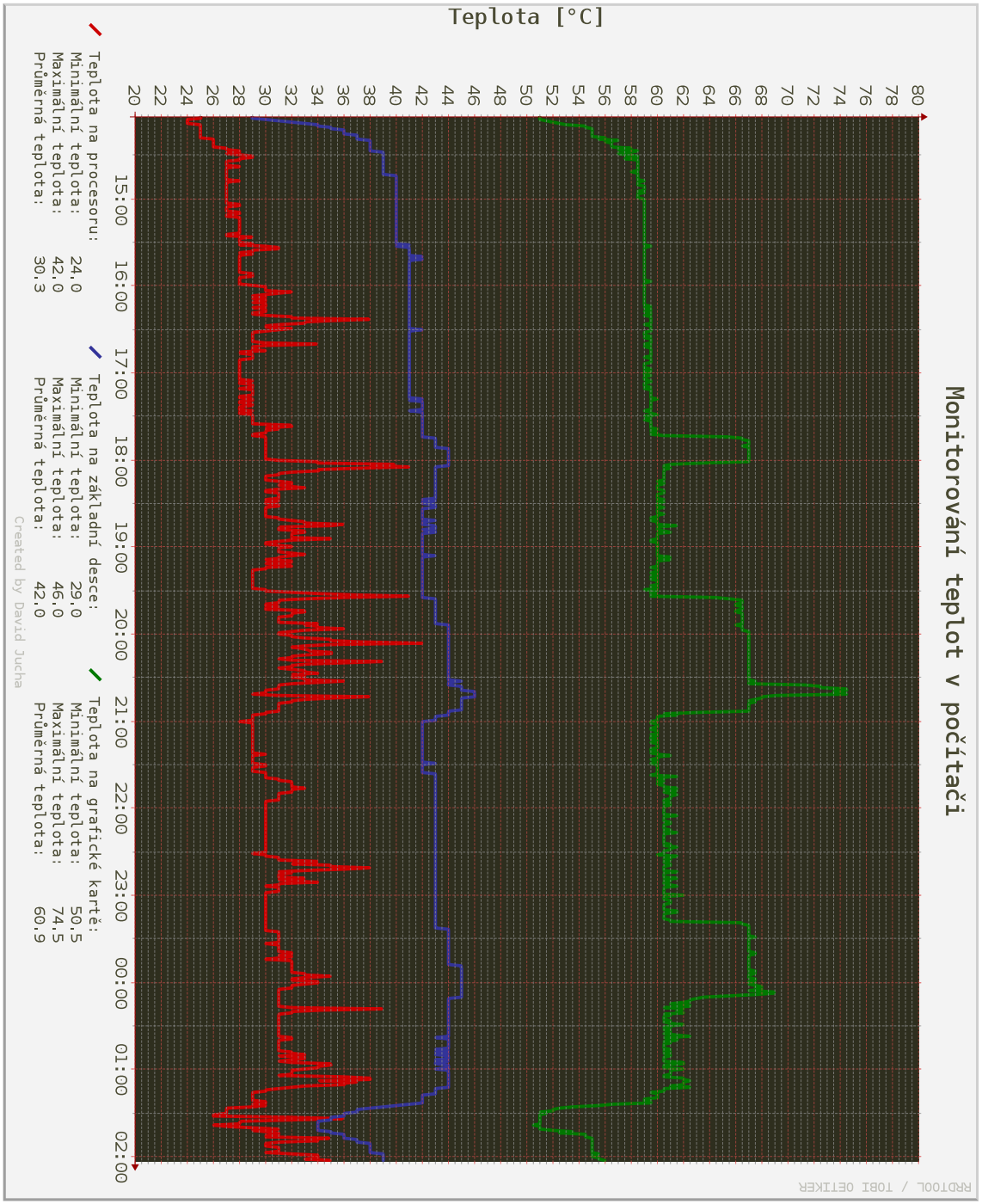


Obrázek D.4: Teplota na procesoru a jeho jádrech za posledních 12 hodin

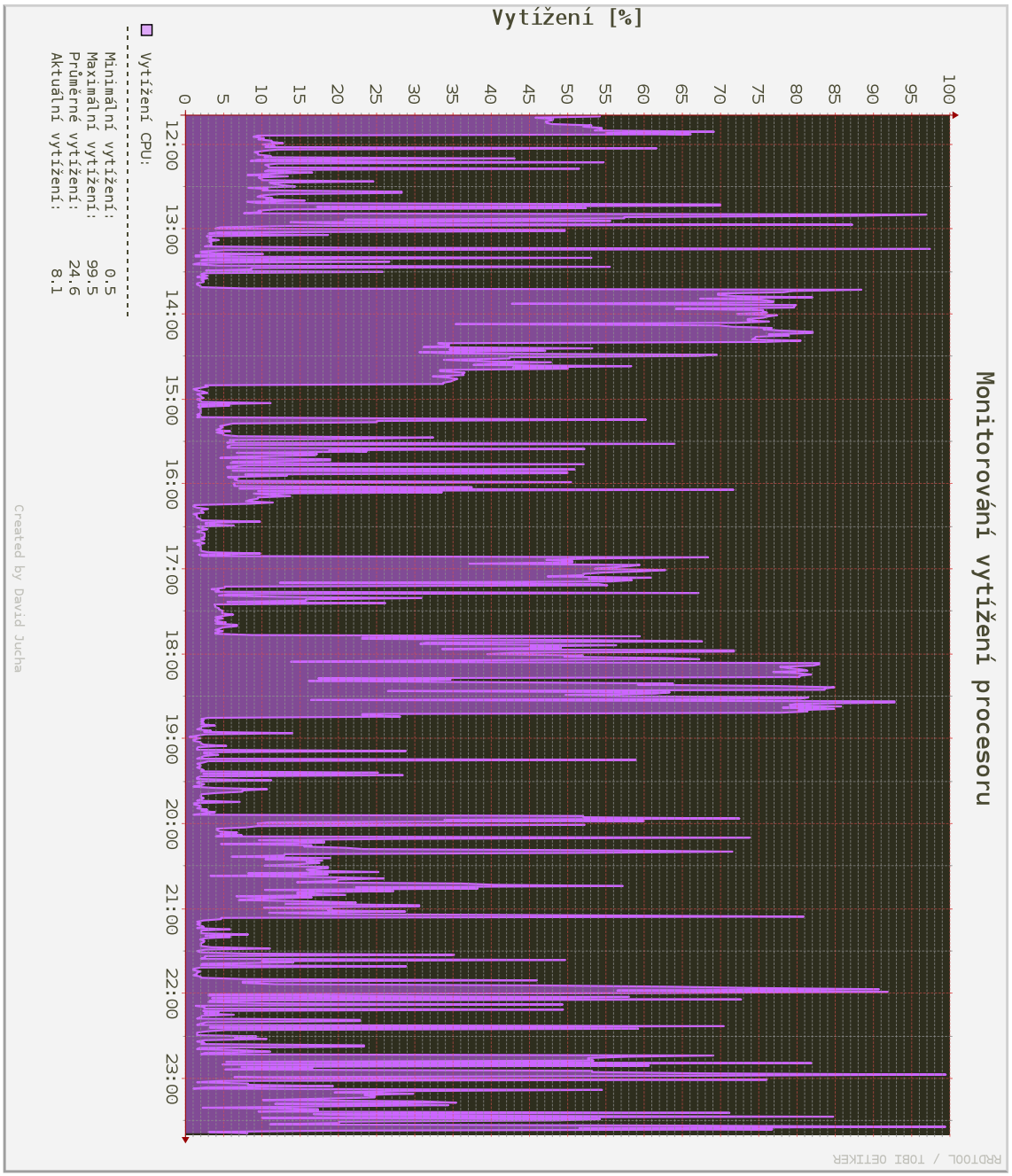


Obrázek D.5: Teplota na grafické kartě za posledních 12 hodin

D SADA GRAFŮ Z PROVEDENÝCH MĚŘENÍ

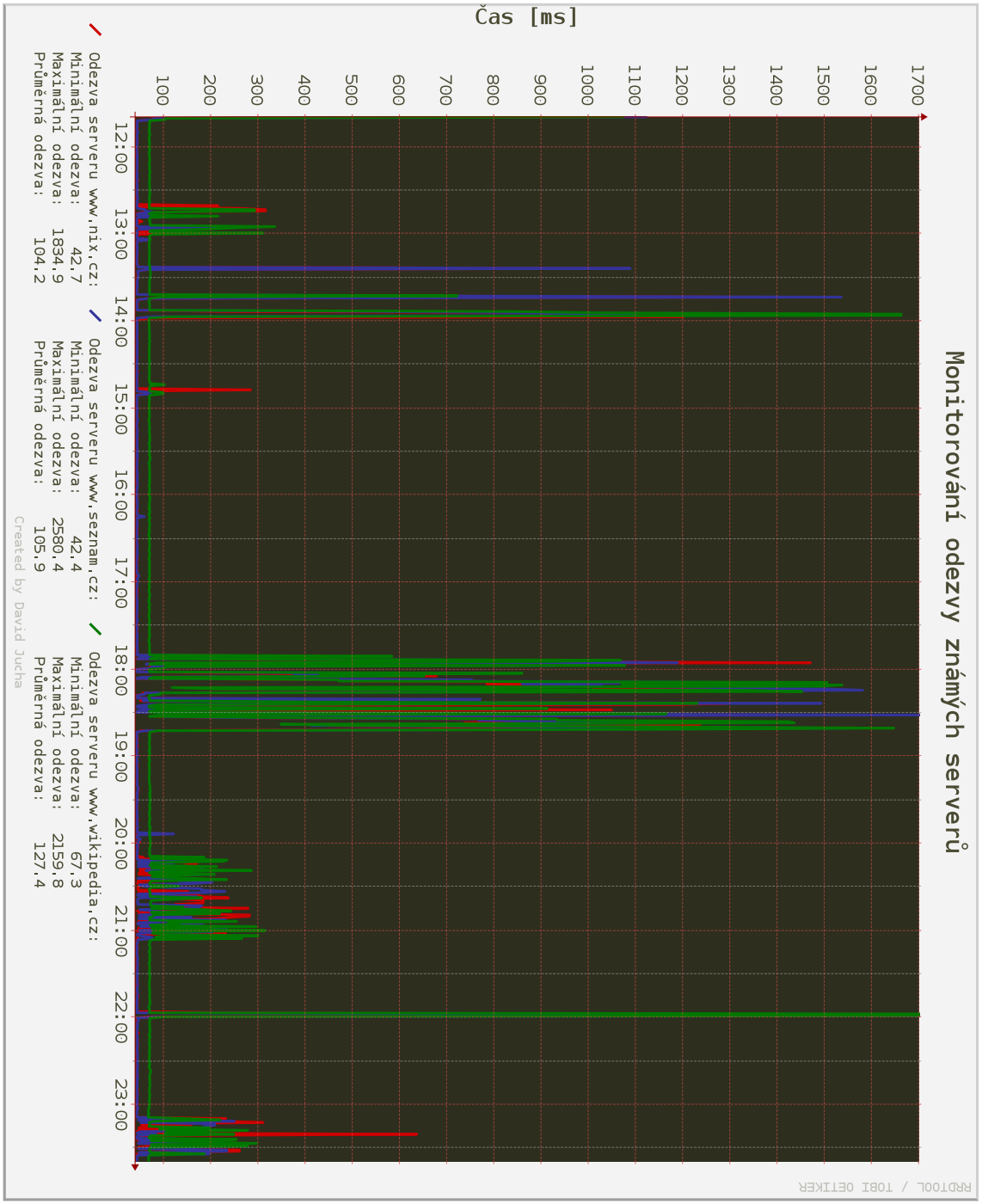


Obrázek D.6: Teploty na základní desce, procesoru a grafické kartě za posledních 12 hodin

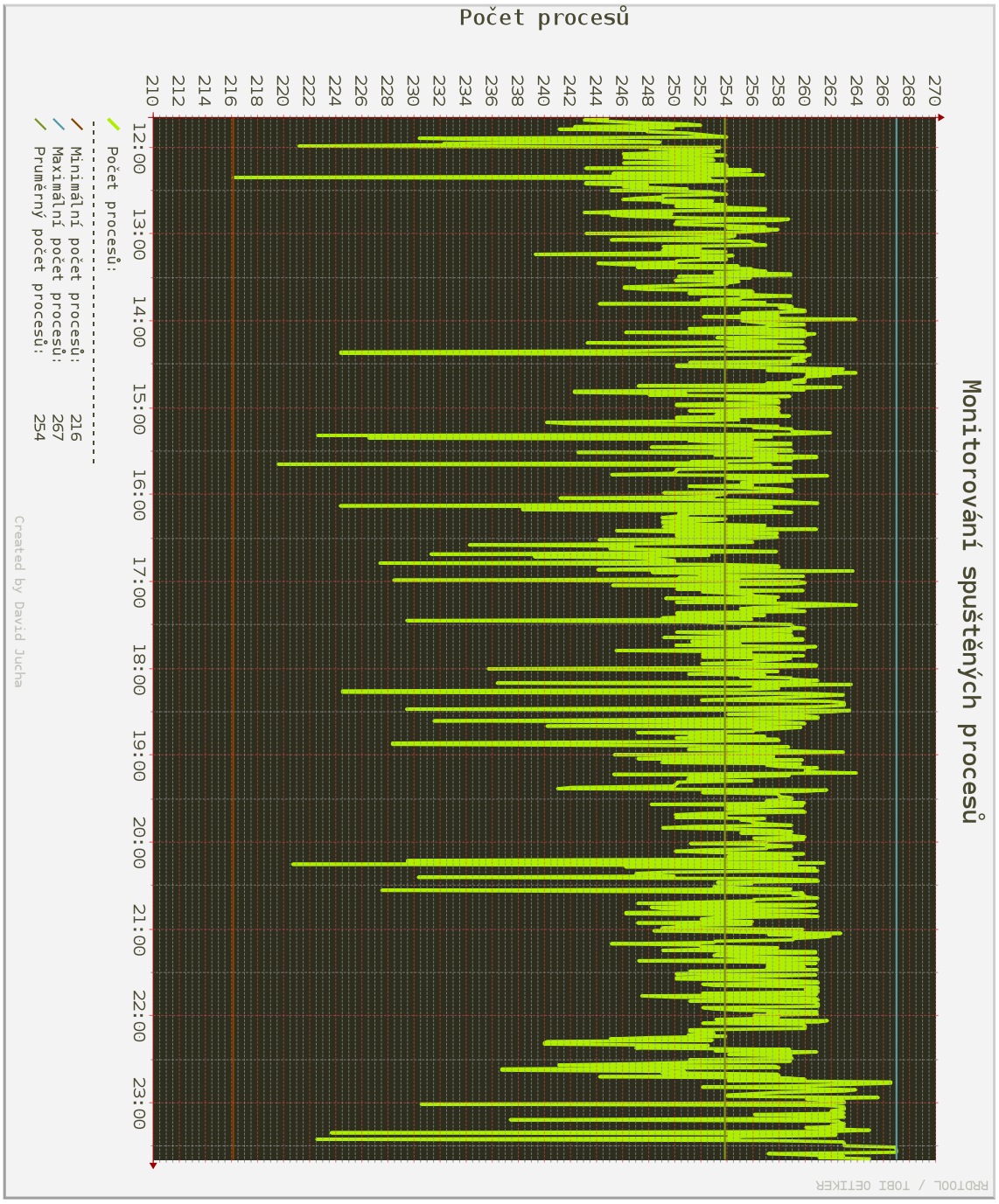


Obrázek D.7: Vytížení procesoru za posledních 12 hodin

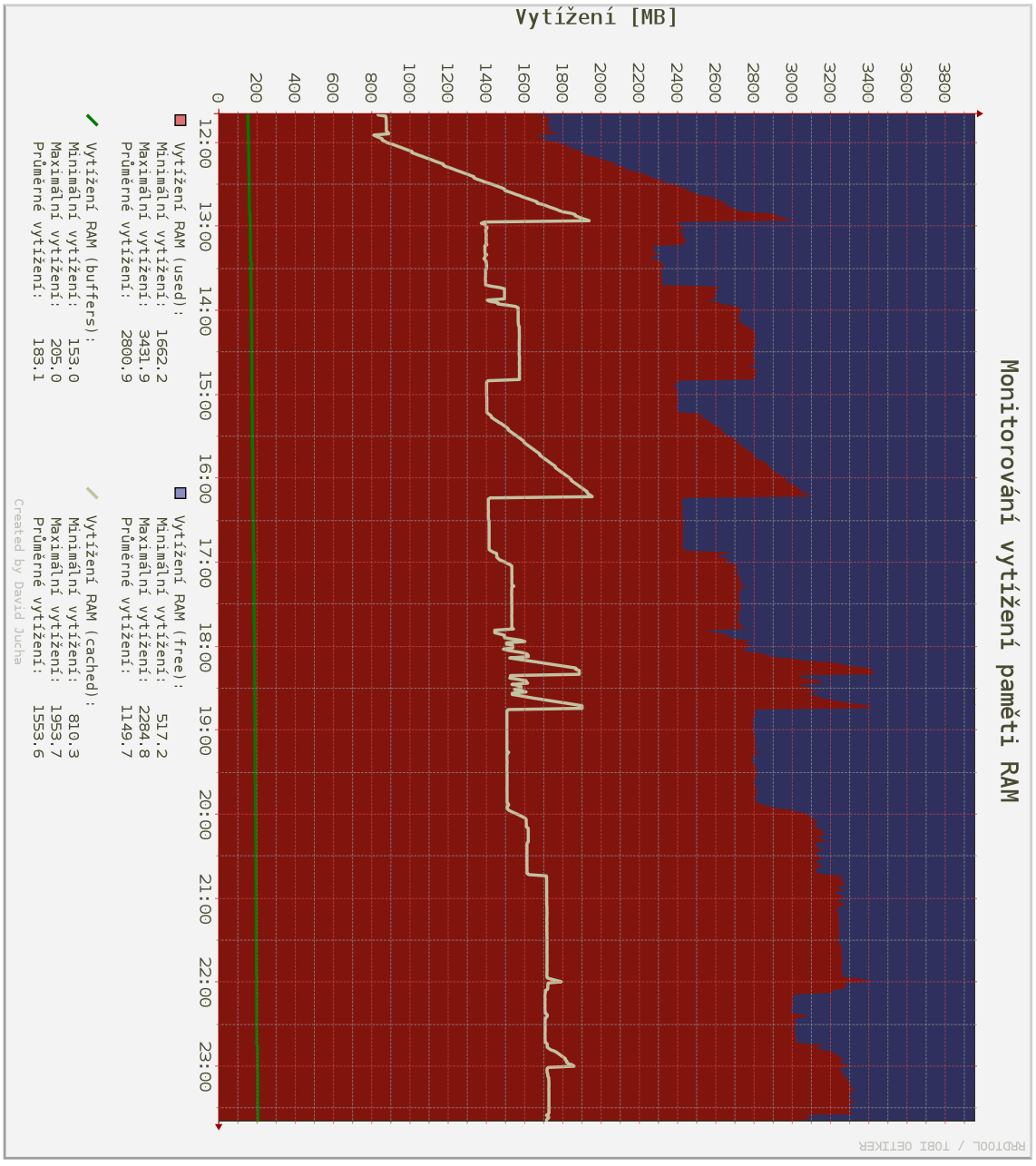
D SADA GRAFŮ Z PROVEDENÝCH MĚŘENÍ



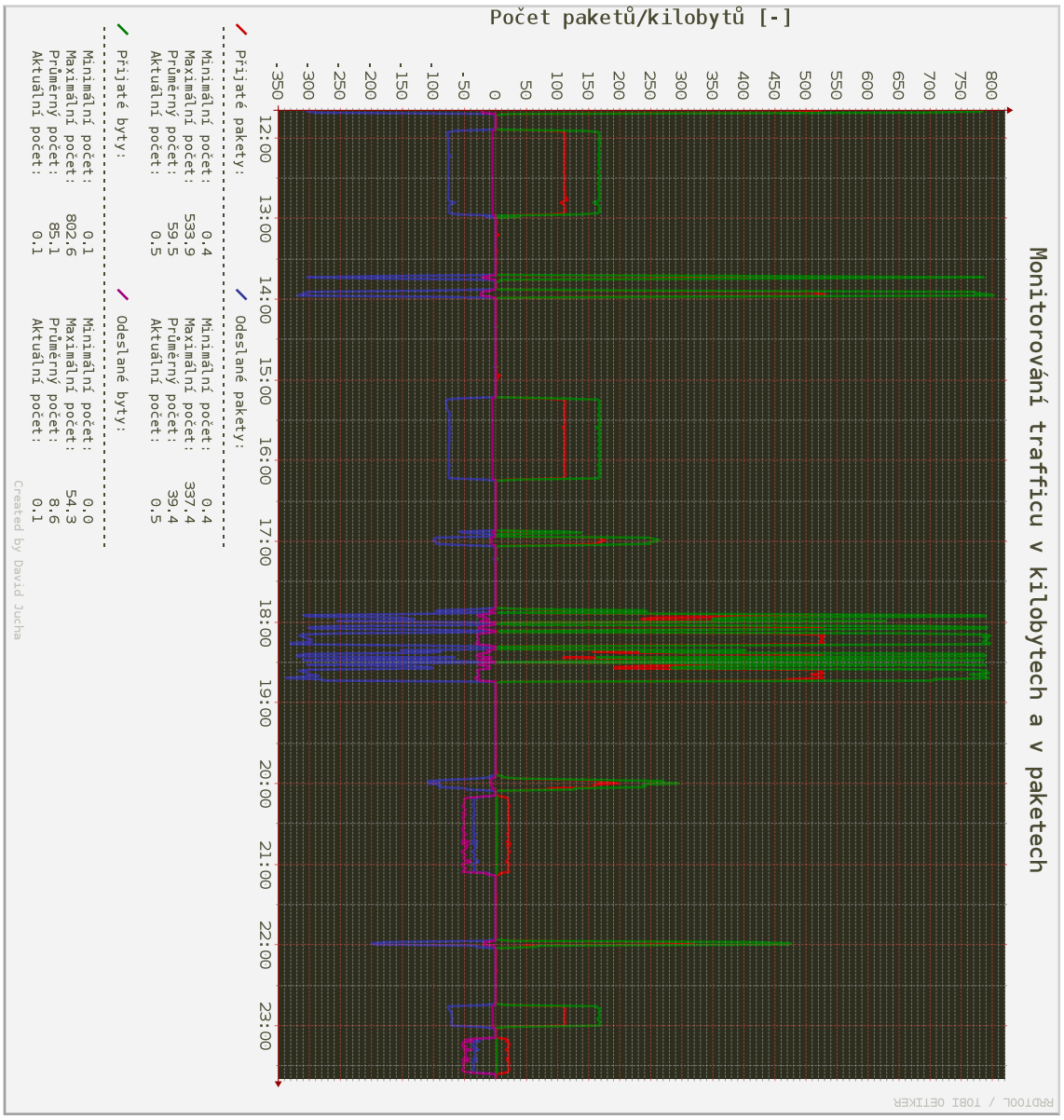
Obrázek D.8: Odezva známých serverů za posledních 12 hodin



Obrázek D.9: Počet spuštěných procesů za posledních 12 hodin

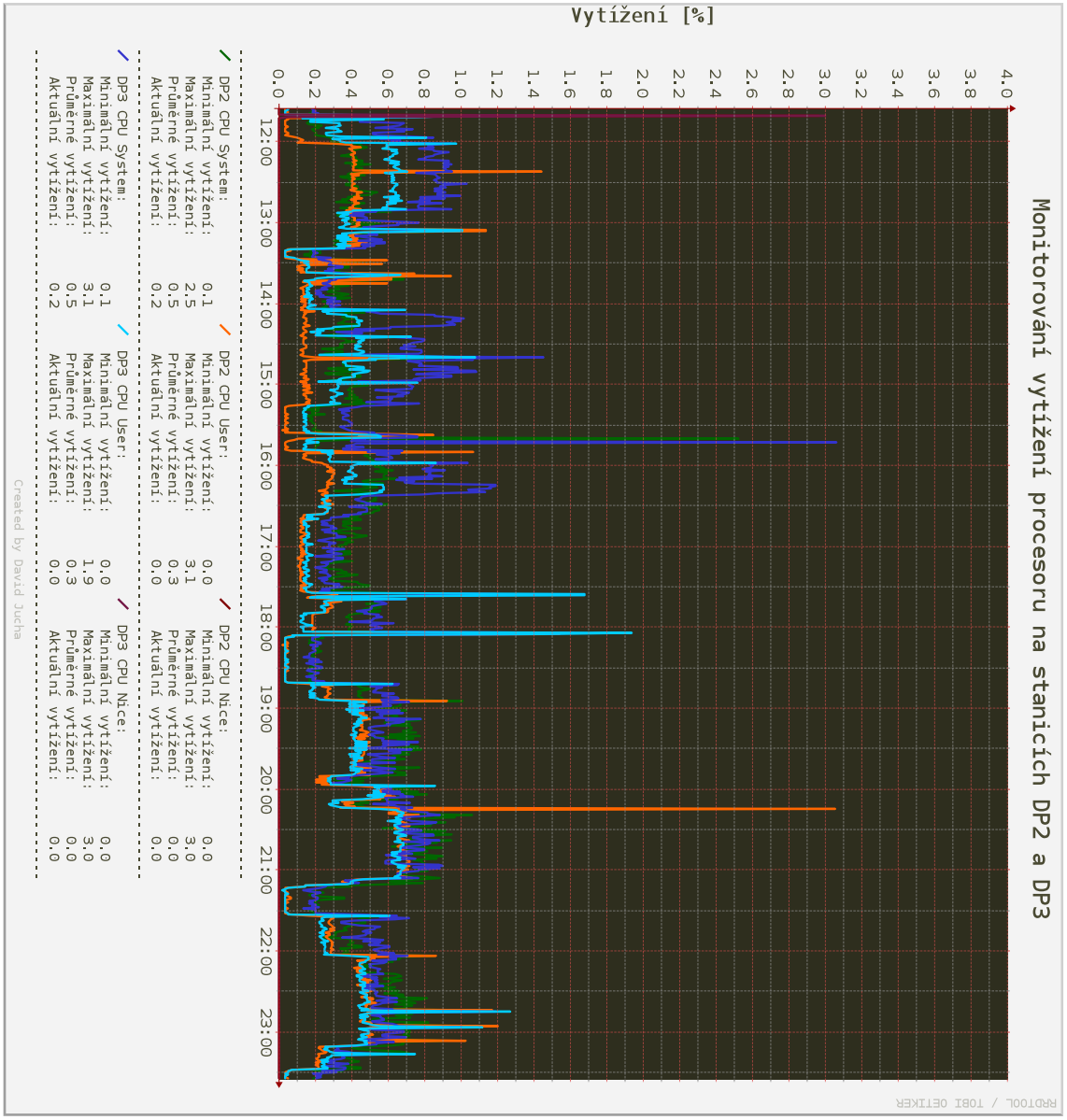


Obrázek D.10: Vytížení paměti RAM za posledních 12 hodin

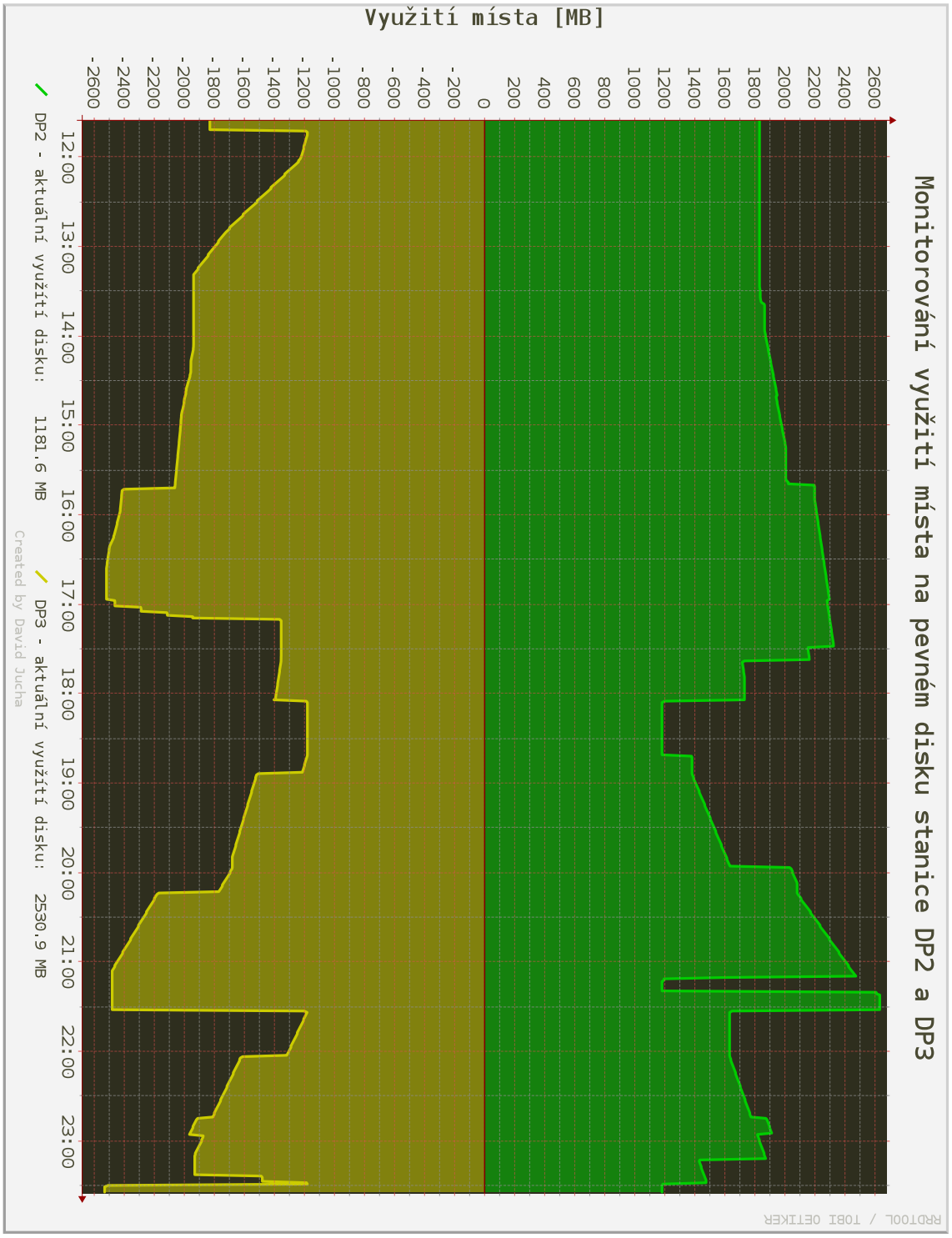


Obrázek D.11: Traffic v obou směrech v kilobytech a v paketech za posledních 12 hodin

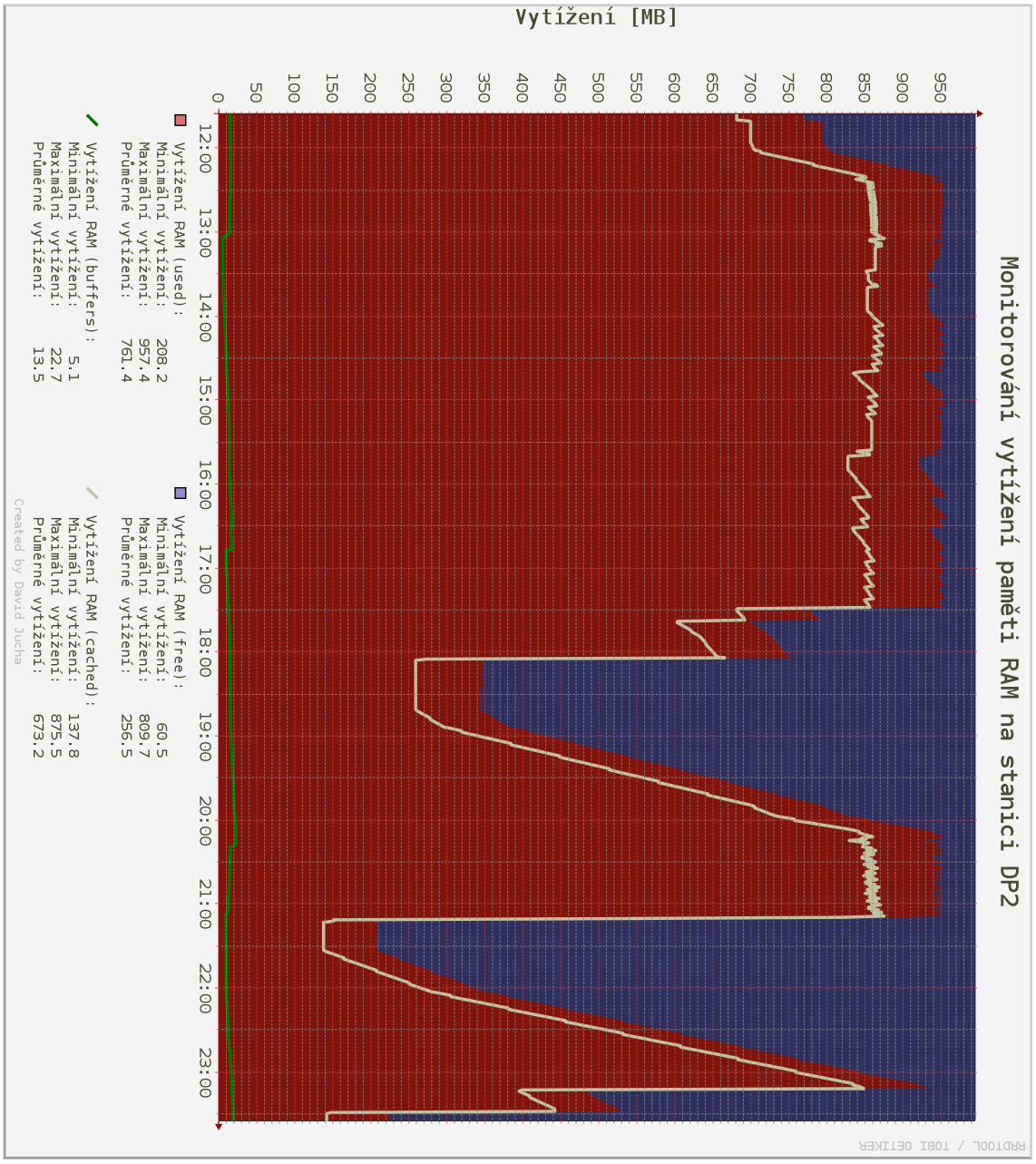
D.2 Sada grafů ze síťových stanic za posledních 12 hodin



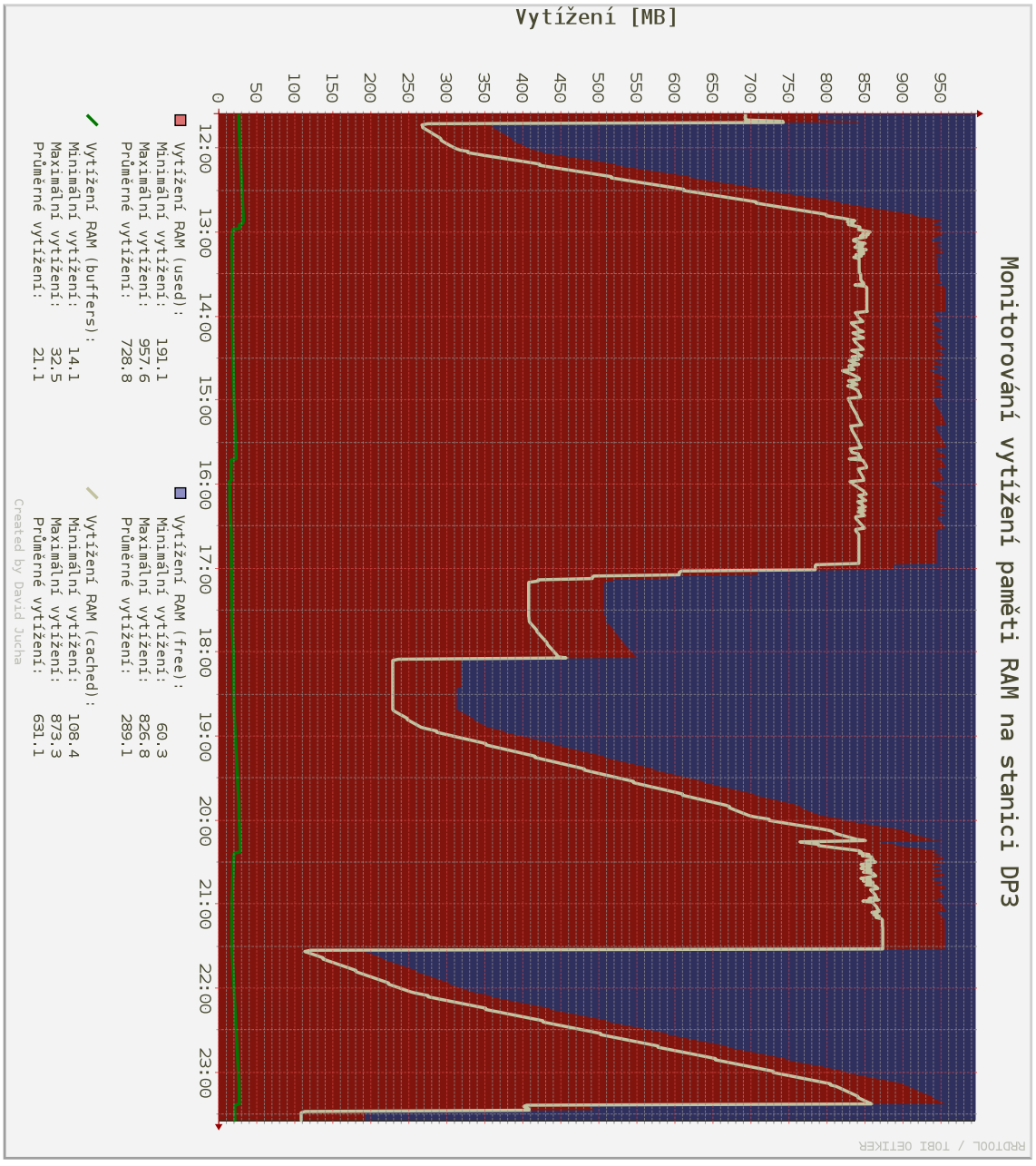
Obrázek D.12: Vytížení procesoru na stanicích DP2 a DP3 za posledních 12 hodin



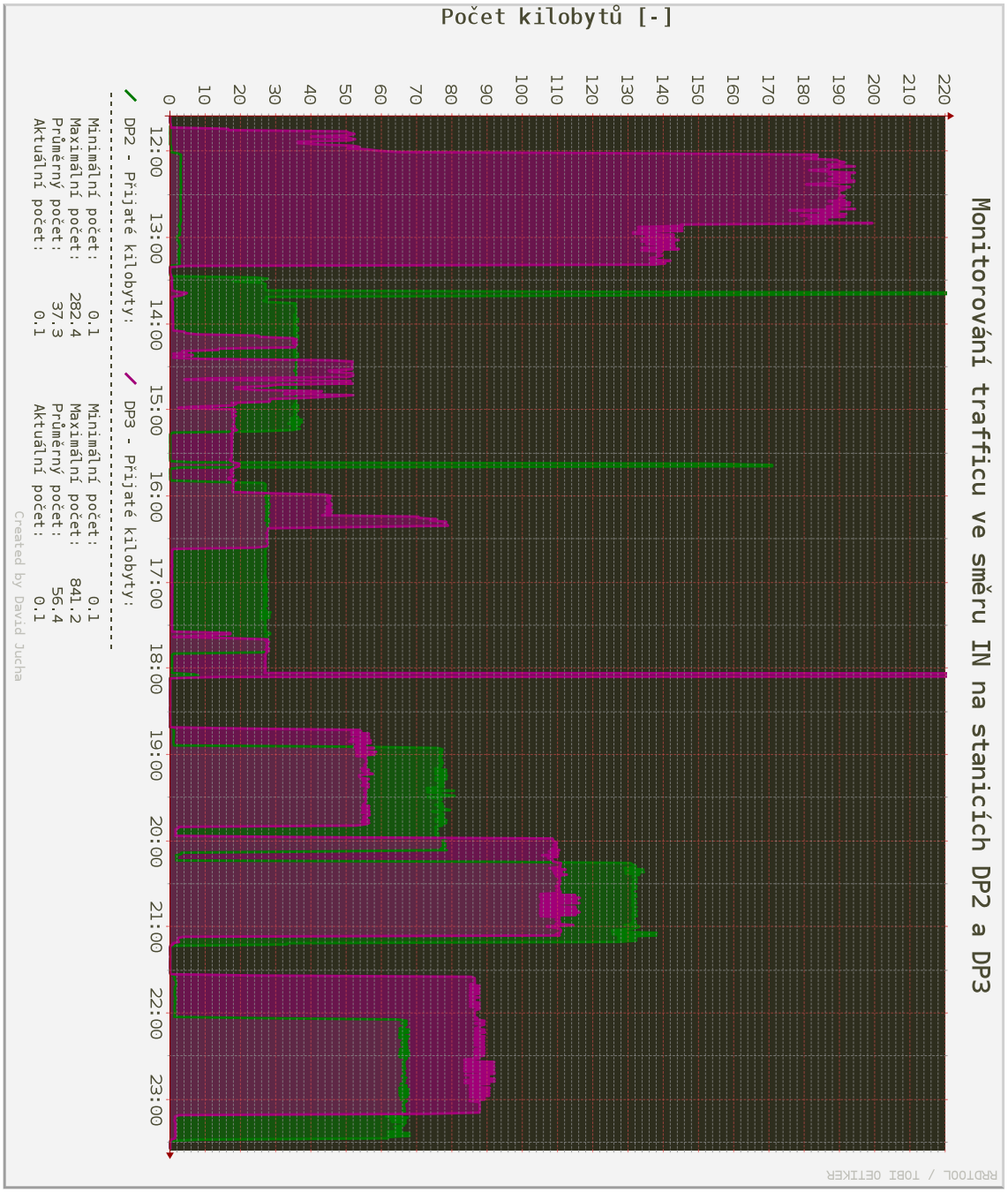
Obrázek D.13: Využití místa na pevném disku stanice DP2 a DP3 za posledních 12 hodin



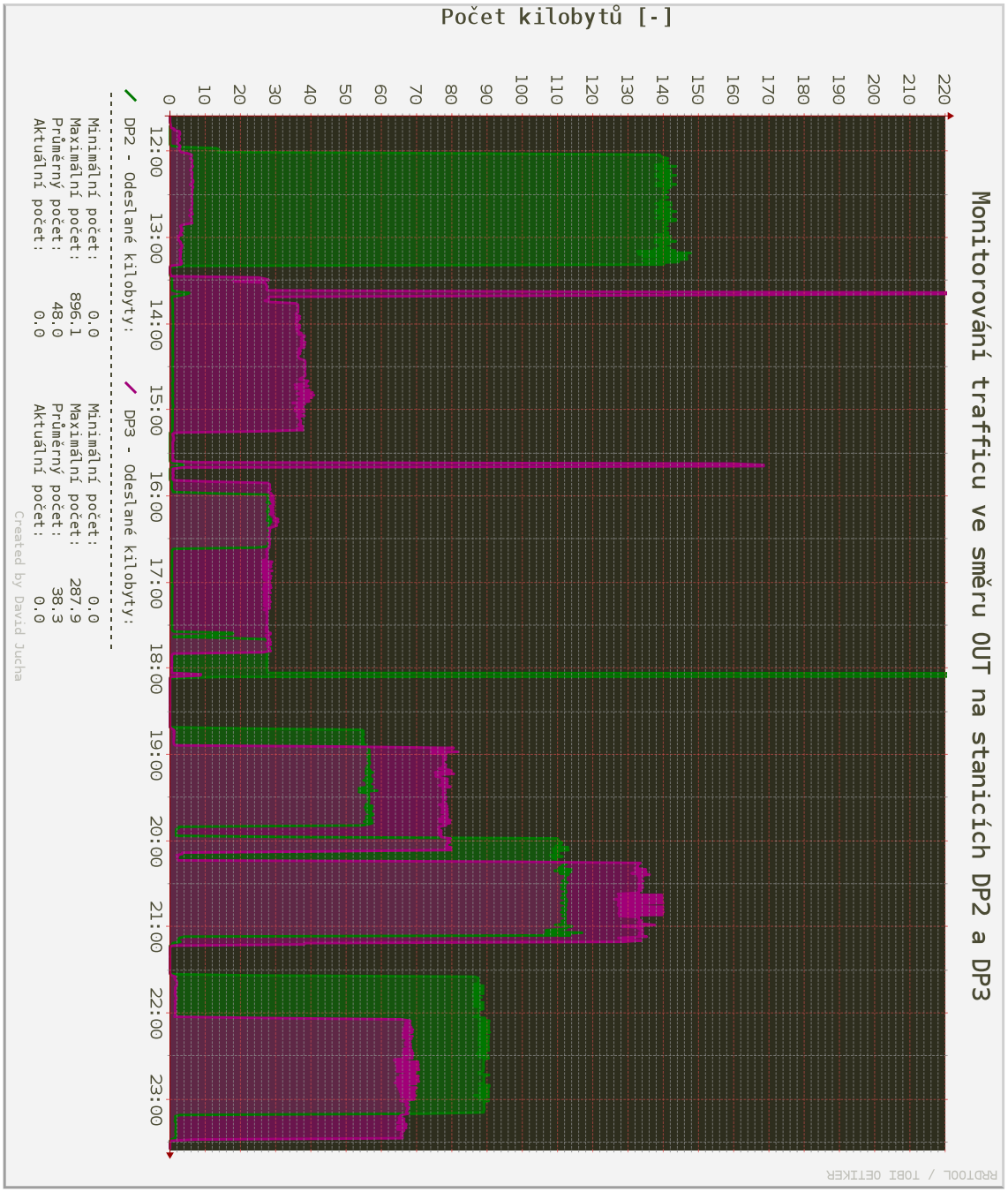
Obrázek D.14: Vytížení paměti RAM na stanici DP2 za posledních 12 hodin



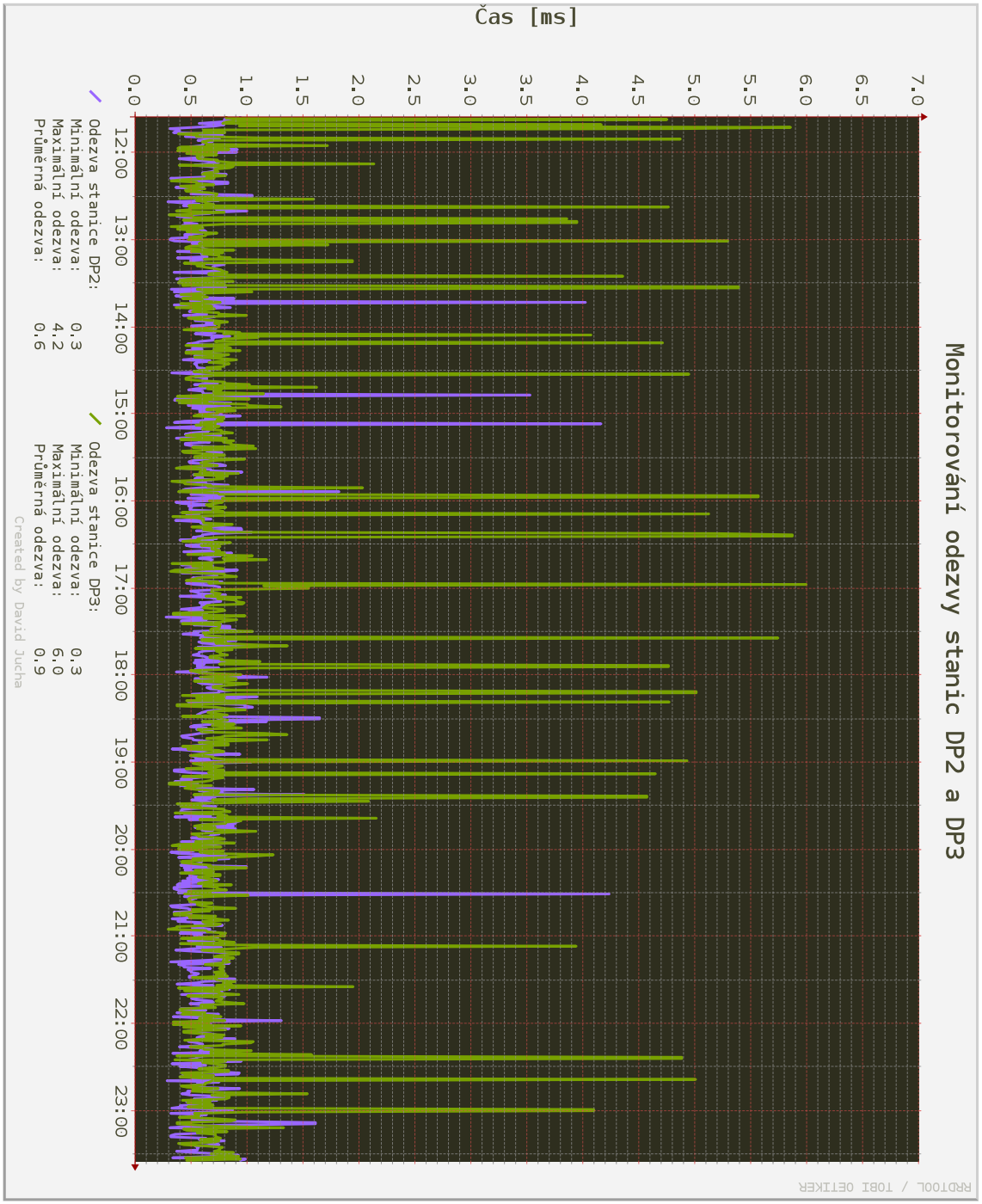
Obrázek D.15: Vytížení paměti RAM na stanici DP3 za posledních 12 hodin



Obrázek D.16: Traffic ve směru IN na stanicích DP2 a DP3 za posledních 12 hodin

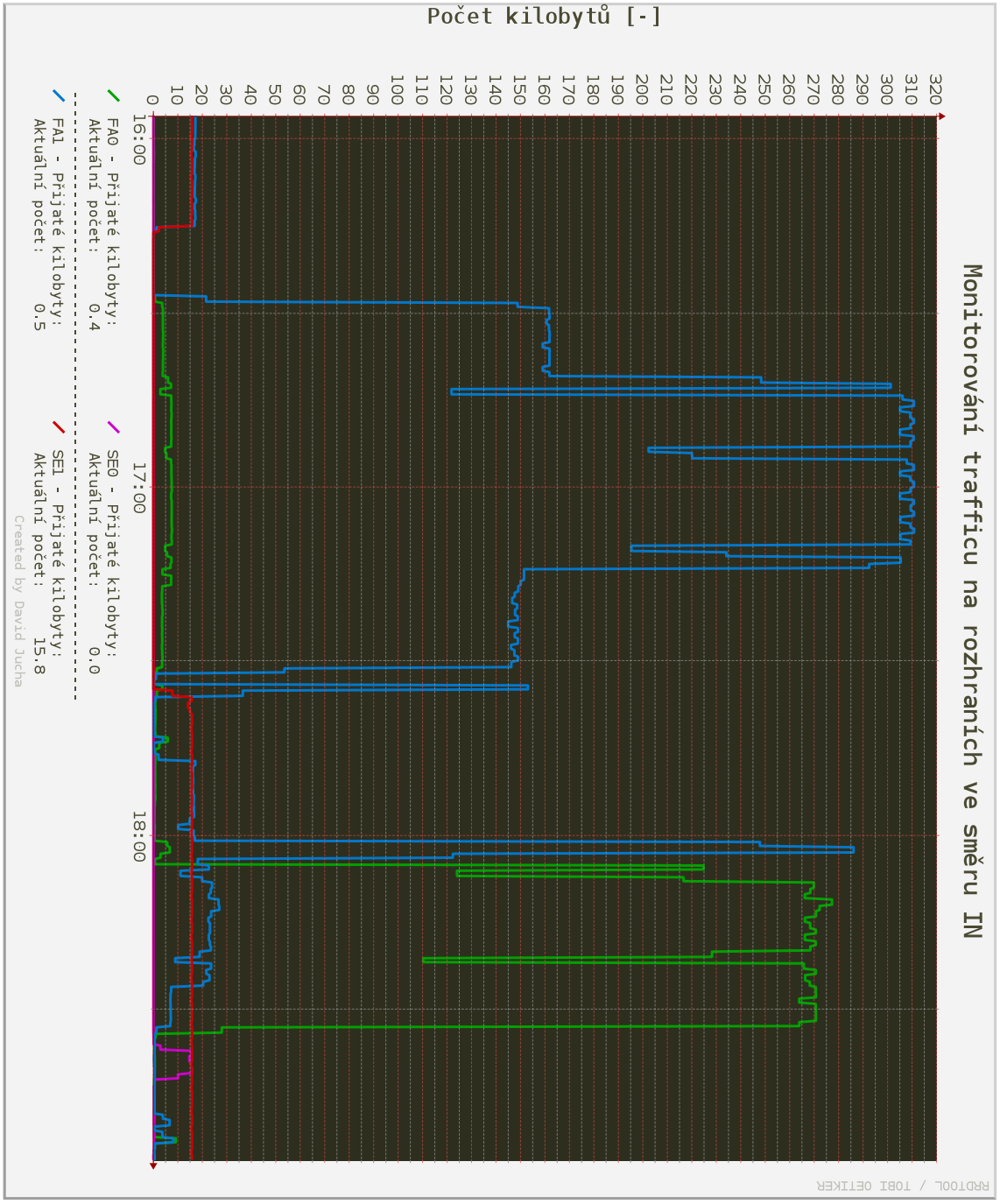


Obrázek D.17: Traffic ve směru OUT na stanicích DP2 a DP3 za posledních 12 hodin

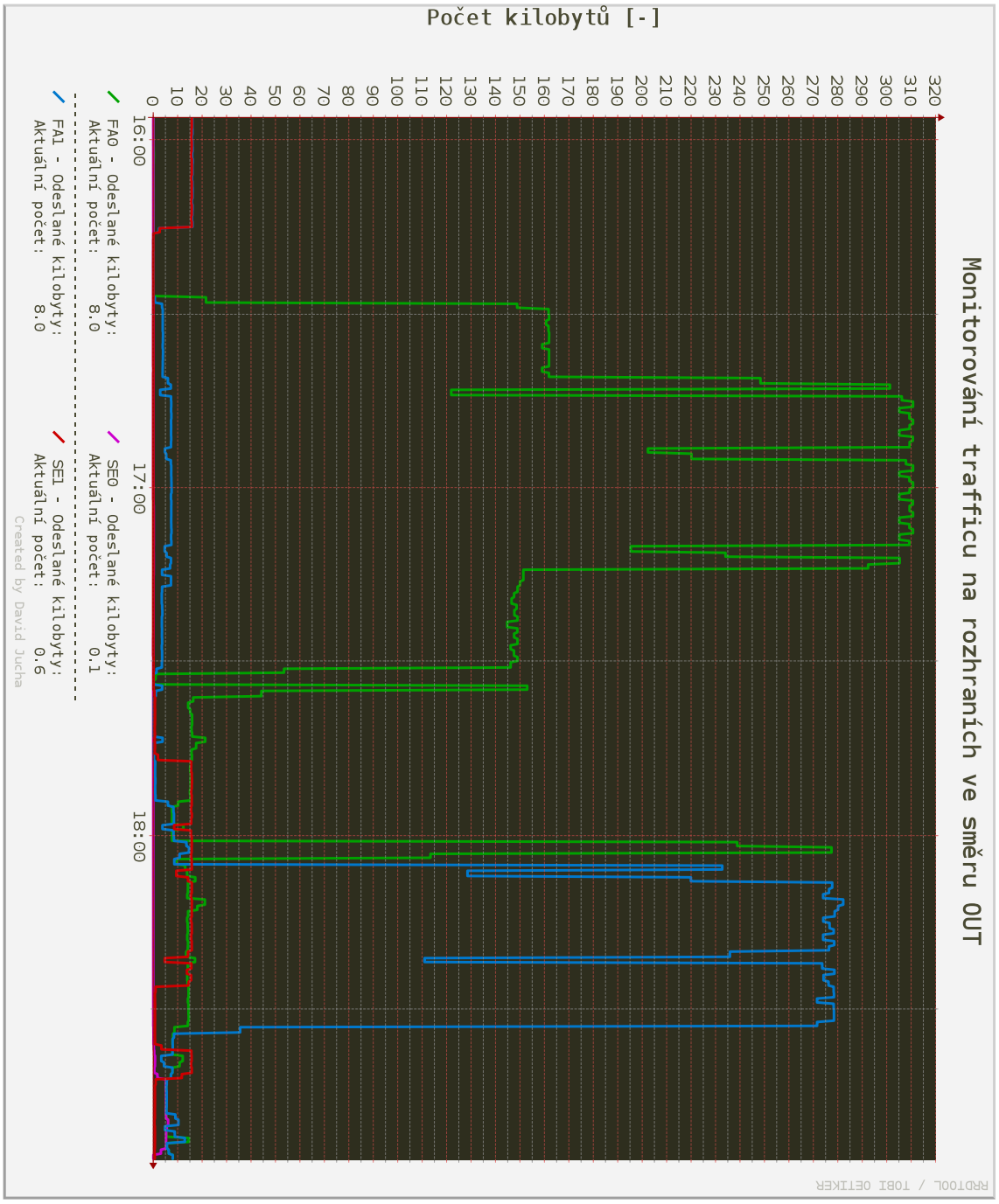


Obrázek D.18: Odezva stanic DP2 a DP3 za posledních 12 hodin

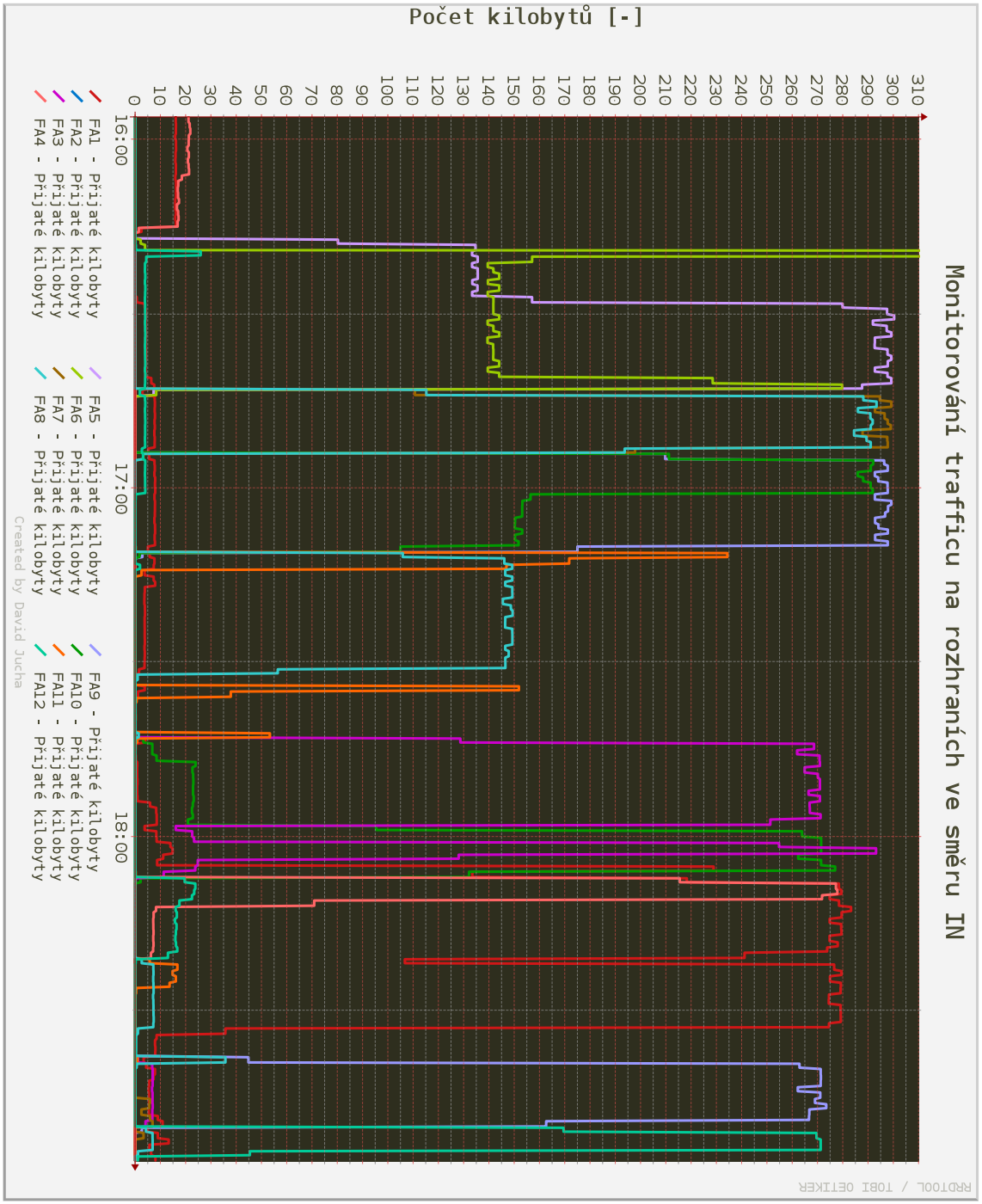
D.3 Sada grafů ze síťových zařízení za poslední 3 hodiny



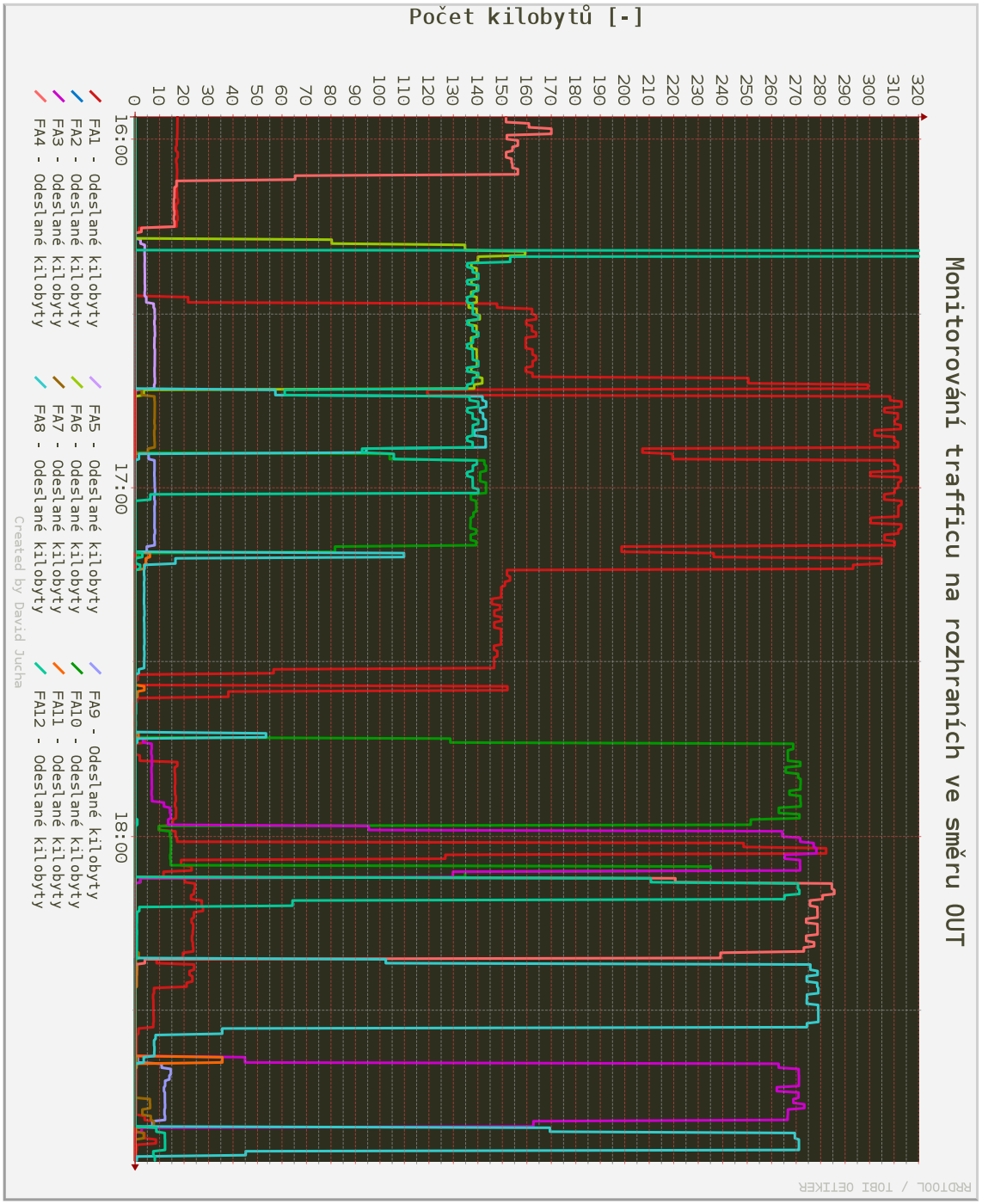
Obrázek D.19: Traffic ve směru IN na všech rozhraních routeru Cisco 2800 za poslední tři hodiny



Obrázek D.20: Traffic ve směru OUT na všech rozhraních routeru Cisco 2800 za poslední tři hodiny



Obrázek D.21: Traffic ve směru IN na všech rozhraních switche Cisco Catalyst 3560 za poslední tři hodiny



Obrázek D.22: Traffic ve směru OUT na všech rozhraních switche Cisco Catalyst 3560 za poslední tři hodiny

E Sada návodů pro praktickou výuku

E.1 Lokální sběr informací z PC, notebooku nebo serveru s využitím nástroje Lm-sensors

V případě, že chceme sbírat informace z dané stanice pomocí nástrojů RRDtool a Lm-sensors, je potřeba dodržet následující kroky:

1. Instalace nástroje RRDtool (viz kapitola 3.1)
2. Instalace SW démona Cron (viz kapitola 2.4.1)
3. Instalace nástroje Lm-sensors a inicializace senzorů (viz kapitola 2.5.1)
4. Vytvoření skriptu pro:
   (a) Vytvoření databáze pomocí funkce CREATE (viz kapitola 3.2), ukázkový skript (viz 3.2.1)
   (b) Vkládání získávaných hodnot do databáze pomocí funkce UPDATE (viz kapitola 3.3), ukázkový skript (viz 3.3.1)
   (c) Vytváření grafů pomocí funkce GRAPH (viz kapitola 3.4), ukázkový skript (viz 3.4.1)
5. Změna práv, aby byl skript spustitelný (viz kapitola 2.4.1)
6. Vložení záznamu s cestou ke skriptu do SW démona Cron (viz kapitola 2.4.1)

Po dodržení těchto šesti kroků se nám začnou vytvářet grafy, které budou umísťovány do složky, definované v námi vytvořeném skritpu.

Všechny připravené skripty odpovídající kapitolám 4.1.1 - 4.1.2 a C.1.1 - C.1.4 jsou umístěny v příloze na CD. Skripty obsahují následující měření:

- Napětí na procesoru (tzv. vcore napětí)
- Napětí na jednotlivých větvích zdroje (+3,3 V; +5 V; +12 V)
- Rychlosti otáček ventilátorů
- Teplota na procesoru a jeho jádrech
- Teplota na grafické kartě ATI Radeon HD4670
- Teploty na základní desce, procesoru a grafické kartě

E.2 Lokální sběr informací z PC, notebooku nebo serveru s využitím vestavěných nástrojů OS Ubuntu

V případě, že chceme sbírat informace z dané stanice pomocí nástroje RRDtool a nástrojů vestavěných v OS Ubuntu, je potřeba dodržet následující kroky:

1. Instalace nástroje RRDtool (viz kapitola 3.1)
2. Instalace SW démona Cron (viz kapitola 2.4.1)
3. Vytvoření skriptu pro:
   (a) Vytvoření databáze pomocí funkce CREATE (viz kapitola 3.2), ukázkový skript (viz 3.2.1)
   (b) Vkládání získávaných hodnot do databáze pomocí funkce UPDATE (viz kapitola 3.3), ukázkový skript (viz 3.3.1)
   (c) Vytváření grafů pomocí funkce GRAPH (viz kapitola 3.4), ukázkový skript (viz 3.4.1)
4. Změna práv, aby byl skript spustitelný (viz kapitola 2.4.1)
5. Vložení záznamu s cestou ke skriptu do SW démona Cron (viz kapitola 2.4.1)

Po dodržení těchto pěti kroků se nám začnou vytvářet grafy, které budou umísťovány do složky, definované v námi vytvořeném skritpu.

Všechny připravené skripty odpovídající kapitolám 4.1.3 - 4.1.4 a C.1.5 - C.1.7 jsou umístěny v příloze na CD. Skripty obsahují následující měření:

- Vytížení procesoru
- Počet spuštěných procesů
- Odezva známých serverů
- Vytížení paměti RAM
- Traffic v obou směrech v kilobytech a v paketech

E.3 Sběr informací z PC, notebooků nebo serverů v počítačové síti

V případě, že chceme sbírat informace pomocí nástroje RRDtool ze stanic, zapojených v síti s využitím protokolu SNMP, je potřeba dodržet následující kroky na straně serveru:

1. Instalace nástroje RRDtool (viz kapitola 3.1)
2. Instalace SW démona Cron (viz kapitola 2.4.1)
3. Instalace a konfigurace protokolu SNMP (viz kapitola 1.6.1)
4. Vytvoření skriptu pro:
   (a) Vytvoření databáze pomocí funkce CREATE (viz kapitola 3.2), ukázkový skript (viz 3.2.1)
   (b) Vkládání získávaných hodnot do databáze pomocí funkce UPDATE (viz kapitola 3.3), ukázkový skript (viz 3.3.1)
   (c) Vytváření grafů pomocí funkce GRAPH (viz kapitola 3.4), ukázkový skript (viz 3.4.1)
5. Změna práv, aby byl skript spustitelný (viz kapitola 2.4.1)
6. Vložení záznamu s cestou ke skriptu do SW démona Cron (viz kapitola 2.4.1)

A na straně klientů:

1. Instalace a konfigurace SNMP démona (viz kapitola 1.6.2)

Po dodržení těchto šesti kroků na straně serveru a jednoho kroku na klientských stanicích se nám začnou vytvářet grafy, které budou umísťovány do složky, definované v námi vytvořeném skritpu.

Všechny připravené skripty odpovídající kapitolám 4.2.1 - 4.2.2 a C.2.1 - C.2.4 jsou umístěny v příloze na CD. Skripty obsahují následující měření:

- Využití místa na pevném disku stanice DP2 a DP3
- Vytížení paměti RAM na stanici DP2 a odděleně i na DP3
- Vytížení procesoru na stanicích DP2 a DP3
- Traffic ve směru IN na stanicích DP2 a DP3
- Traffic ve směru OUT na stanicích DP2 a DP3
- Odezva stanic DP2 a DP3

E.4 Sběr informací z jiných zařízení v počítačové síti

V případě, že chceme sbírat informace pomocí nástroje RRDtool z jiných zařízení, umístěných v počítačové síti (zde se jedná o routery a switche Cisco) s využitím protokolu SNMP, je potřeba dodržet následující kroky na straně serveru:

1. Instalace nástroje RRDtool (viz kapitola 3.1)
2. Instalace SW démona Cron (viz kapitola 2.4.1)
3. Instalace a konfigurace protokolu SNMP (viz kapitola 1.6.1)
4. Vytvoření skriptu pro:
   (a) Vytvoření databáze pomocí funkce CREATE (viz kapitola 3.2), ukázkový skript (viz 3.2.1)
   (b) Vkládání získávaných hodnot do databáze pomocí funkce UPDATE (viz kapitola 3.3), ukázkový skript (viz 3.3.1)
   (c) Vytváření grafů pomocí funkce GRAPH (viz kapitola 3.4), ukázkový skript (viz 3.4.1)
5. Změna práv, aby byl skript spustitelný (viz kapitola 2.4.1)
6. Vložení záznamu s cestou ke skriptu do SW démona Cron (viz kapitola 2.4.1)

Na straně klienta - routeru Cisco 2800:

1. Instalace a konfigurace protokolu SNMP (viz kapitola 4.3.1)

A na straně klienta - switche Cisco Catalyst 3560:

1. Instalace a konfigurace protokolu SNMP (viz kapitola 4.3.2)

Po dodržení těchto šesti kroků na straně serveru a vždy po jednom kroku na routeru nebo switchi se nám začnou vytvářet grafy, které budou umísťovány do složky, definované v námi vytvořeném skritpu.

Všechny připravené skripty odpovídající kapitolám 4.3.3 - 4.3.4 a C.3.1 - C.3.2 jsou umístěny v příloze na CD. Skripty obsahují následující měření:

- Traffic ve směru IN na všech rozhraních routeru Cisco 2800
- Traffic ve směru OUT na všech rozhraních routeru Cisco 2800
- Traffic ve směru IN na rozhraních switche Cisco Catalyst 3560
- Traffic ve směru OUT na rozhraních switche Cisco Catalyst 3560